Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Uma Linguagem Específica do Domínio para uma abordagem Orientada aos Objectivos baseada em KAOS

Ana Cristina de Freitas Dias (26434)

1º Semestre de 2008/09

20 de Fevereiro de 2009

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Uma Linguagem Específica do Domínio para uma abordagem Orientada aos Objectivos baseada em KAOS

Ana Cristina de Freitas Dias (26434)

Orientador: Prof. Doutor Vasco Miguel Moreira do Amaral
Co-orientador: Prof. Doutor João Baptista da Silva Araújo Junior

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

1º Semestre de 2008/09

20 de Fevereiro de 2009

Agradecimentos

Queria agradecer aos que, de uma maneira ou de outra, me ajudaram na realização deste trabalho.

Primeiro que tudo gostaria de agradecer aos meus orientadores, os professores Vasco Amaral e João Araújo, pelas suas sugestões, as suas críticas e todo o trabalho, disponibilidade e paciência mostrados para que pudesse concluir este trabalho. Gostaria também de deixar uma palavra de apreço à professora Carla Silva da Universidade Federal de Pernambuco, no Brasil, pelas suas sugestões e pelo interesse e disponibilidade em ajudar, nomeadamente na fase de testes à ferramenta. Agradeço também ao meu colega e amigo Carlos Nunes, pela inesgotável paciência e disponibilidade em me ajudar no desenvolvimento da ferramenta criada com este trabalho.

Quero também agradecer aos meus amigos que, de uma forma ou de outra, me ajudaram durante o tempo que durou este trabalho, não só pelo interesse demonstrado, apoio moral, troca de ideias, como também fazerem-me rir, fazerem companhia à hora do almoço ou pelas "discussões intermináveis" sobre quais os melhores computadores do mercado ou a confusão inerente às grandes cidades.

No final, mas por isso não menos importantes, aos meus pais e à minha irmã, que se ofereceram para rever o relatório apesar de não serem da área, darem as suas opiniões para melhorar o texto, ou então simplesmente porque estiveram sempre lá quando precisei deles.

Resumo

A **E**ngenharia de **R**equisitos (ER) é o ramo da Engenharia de Software que lida com a identificação, análise, especificação e teste de requisitos para sistemas de software. Um requisito de software é uma propriedade que deve ser exibida pelo software desenvolvido ou adaptado para resolver um determinado problema. Dentro da Engenharia de Requisitos, existem vários ramos de metodologias para obter requisitos, entre os quais a **E**ngenharia de **R**equisitos **O**rientada a **O**bjectivos (EROO), que usa objectivos para elicitar, desenvolver, estruturar, especificar, analisar, negociar, documentar e modificar requisitos.

A **M**odelação **E**specífica do **D**omínio (MED) aumenta o nível de abstracção de uma solução através da utilização de conceitos do domínio em análise. Este tipo de modelação aumenta muito a produtividade pois cada símbolo do modelo corresponde a um conceito do domínio que por sua vez corresponde a um conjunto de linhas de código específico. O problema do domínio pode ser modelado com a recurso a uma **L**inguagem **E**specífica do **D**omínio (LED).

A complexidade visual de diagramas EROO padrão pode ficar muito grande devido ao elevado número de objectivos a serem refinados e detalhados nos modelos. Este problema acontece tipicamente em sistemas reais devido à sua complexidade inerente podendo torná-los ilegíveis e difíceis de gerir e, como consequência, os modelos podem tornar-se mais difíceis de validar e actualizar. Assim, esta dissertação propõe uma extensão a uma linguagem EROO pela introdução do conceito de *Compartimento*, uma técnica de encapsulamento para guardar os conceitos e com possibilidade de lidar com técnicas de interface com o utilizador, como a colapsação das caixas que representam os Compartimentos, com o propósito de melhorar a escalabilidade dos modelos. As ferramentas também não verificam a sintaxe dos modelos, o que pode provocar a inconsistência nos mesmos.

Para desenvolver a ferramenta foi usada a framework Eclipse (com plugins GMF/EMF). Foi escolhida uma metodologia específica EROO chamada KAOS e baseado nisto foi desenhada uma nova LED através da criação do seu meta modelo estendido.

Palavras-chave: KAOS, Modelação Específica do Domínio, EMF/GMF, Engenharia de Requisitos

Abstract

Requirements **E**ngineering (RE) is the branch of Software Engineering dealing with identification, analysis, specification and testing of requirements for software systems. A software requirement is a property which must be exhibited by software developed or adapted to solve a particular problem. Within RE, there are several branches of methodologies for obtaining requirements, among which we have **G**oal-**O**riented **R**equirements **E**ngineering (GORE), that uses goals to elicit, develop, structure, specify, analyze, negotiate, document and modify requirements.

Domain **S**pecific **M**odeling (DSM) raises the level of abstraction of a solution through the use of concepts from the domain in analysis. This kind of modeling increases productivity a lot because each symbol of the model corresponds to a concept of the domain which corresponds to a specific set of lines of code. The domain problem can be modeled with a **D**omain **S**pecific **L**anguage (DSL).

The visual complexity of standard GORE diagrams can get very high due to the high number of goals to be refined and detailed in the models. This problem tipically occurs in real systems due to their complexity which can make them unreadable and difficult to manage and, as a consequence, the models can became harder to validate or update. Thus, this dissertation proposes an extension to a GORE language in order to incorporate the notion of *Compartment*, an encapsulation technique to keep the concepts on the diagrams inside of them and posssibility to handle with user interface techniques, such as colapsing ability of the box representing Compartments with the purpose of improving the scalability of the models. The analised tools do not verify also the syntax of the models, which can cause their inconsistency.

For the making of the tool it was used the Eclipse framework (with GMF/EMF plugins). We have chosen a specific GORE technology named KAOS and based on it we designed a new DSL by creating its extended meta model.

Keywords: KAOS, Domain-Specific Modeling, EMF/GMF, Requirements Engineering

Conteúdo

Lista de Figuras

Lista de Tabelas

1 . Introdução

1.1 Engenharia de Requisitos e Linguagens Específicas de Domínio

De acordo com [34], a Engenharia de Requisitos é o ramo da Engenharia de Software que lida com a elicitação, refinamento e análise de requisitos de sistemas de software. A Engenharia de Requisitos envolve um conjunto de tarefas que ajudam a explicitar o que o utilizador deseja, como vai interagir com o sistema e qual o impacto do sistema no negócio. Os participantes numa actividade de Engenharia de Requisitos são os engenheiros de software e as partes interessadas do projecto. Esta actividade é importante porque para se desenhar um sistema completo que vá de encontro aos desejos do cliente é necessário entender previamente o que ele realmente deseja. O produto final é um documento, disponibilizado para todas as partes (partes interessadas (*stakeholders*) e engenheiros), onde é descrita a especificação dos requisitos indicados pelas partes interessadas. No final a especificação é validada com o cliente e os utilizadores finais para garantir que o que era desejado é apreendido pelas funcionalidades do sistema.

Os principais problemas da Engenharia de Requisitos são [34]: (i) **problemas de âmbito**: limites do sistema mal-definidos ou especificação confusa e desnecessariamente detalhada por parte dos clientes/utilizadores; (ii) **problemas de compreensão**: os clientes/utilizadores nem sempre sabem o que realmente desejam, não têm um bom conhecimento do domínio do problema, pouca compreensão do meio tecnológico, não transmitem bem aquilo que realmente querem ou exprimem requisitos que entram em conflito com requisitos de outras partes interessadas; (iii) **problemas de volatilidade**: os requisitos não são estáticos, mudam ao longo do tempo.

Existem diversos métodos para elicitação de requisitos, como por exemplo:

- Viewpoints[36]: aqui considera-se que toda a informação sobre o sistema não pode ser considerada apenas sobre um ponto de vista. Assim, os requisitos são identificados e organizados de acordo com diferentes *viewpoints*, onde *viewpoint* é encapsulação parcial de informação de um requisito do sistema.

- Aspectos[35]: aqui pretende-se identificar e especificar os *crosscuting concerns*, também chamados de *aspectos*, separadamente de maneira a incentivar a modularização para diminuir os custos de desenvolvimento, manutenção e evolução. Um *crosscuting concern* é parte de um programa que afecta (*crosscuts*) outro *concern*[1].

- Objectos[25]: aqui pretende-se encapsular toda a informação sobre o processo, o produto e a sua funcionalidade dentro de um objecto de requisito.

- Objectivos: aqui os requisitos do sistema são modelados com recurso a *objectivos*, onde

[1] Conjunto de comportamentos necessários por um programa de computador. Estes comportamentos podem ser separados em secções lógicas que permitem que os programas sejam modularizados [1].

os objectivos são especificações que o sistema deve cumprir (para mais informações, ver capítulo 2).

Para este trabalho foi considerado um método de obtenção de requisitos orientado a objectivos, o KAOS.

A **M**odelação **E**specífica do **D**omínio (MED) é um método da área de Engenharia de Software usada para desenhar e desenvolver sistemas. Faz uso sistemático de **L**inguagens **E**specíficas do **D**omínio (LED) para representar as várias facetas do sistema. Estas linguagens têm níveis de abstracção mais altos que as **L**inguagens de **D**omínios **G**erais (LDG), levando a que seja necessário menos esforço e menos detalhes de baixo nível para especificar um dado sistema. Os modelos de requisitos em geral podem ser descritos através das Linguagens Específicas do Domínio.

1.2 Descrição do problema

O KAOS é uma metodologia muito conhecida e estudada na comunidade de Engenharia de Requisitos. Contudo as especificações existentes não são rigorosas e a linguagem não consegue endereçar efectivamente a escalabilidade dos modelos produzidos. Esta situação vai-se repercutir no rigor dos modelos existentes que poderão conter ambiguidades, sendo por isso interpretados de diferentes maneiras. Poderão também acontecer inconsistência nas implementações existentes.

Os modelos construídos com métodos EROO podem-se tornar muito complexos visualmente à medida que o número de elementos presentes no diagrama aumenta. Quando esta situação acontece os modelos podem-se tornar difíceis de ler e compreender e trazer problemas de escalabilidade. Existem ferramentas para modelar diagramas KAOS, tais como a Dia [2] e a Objectiver [11]. Contudo estas não respondem com eficiência ao problema da escalabilidade e complexidade visual de modelos quando existem muitos elementos. Também não verificam a consistência dos modelos criados de acordo com a sua sintaxe e podem ter comportamento inconsistente e instável.

1.3 Solução Apresentada

Para lidar com o problema da consistência e rigor dos modelos é proposto um novo meta modelo da linguagem KAOS, baseado num previamente existente, em que foram "refinados" alguns conceitos já existentes no método (por exemplo, foi inserido o conceito de Requisito Não-Funcional no meta modelo, como uma especialização de um Objectivo, neste caso um Objectivo com características relacionadas com os atributos de qualidade de um projecto). São também propostas ligações mais consistentes e específicas entre modelos.

Para lidar com o problema da escalabilidade de modelos muito grandes, é proposto o conceito de *Compartimento*. Isto é implementado estendendo o meta modelo da linguagem com

umas caixas (Compartimentos) onde é possível guardar os elementos possíveis de colocar nos diagramas. Estas caixas têm poder de colapsação, isto é, é possível com um clique nestas apresentar ou omitir porções do diagrama, de acordo com os elementos que foram inseridos nos Compartimentos. Isto será feito com recurso a uma LED para criar modelos KAOS, com um editor associado, que vai ser implementado a partir de um meta modelo da linguagem KAOS, criado a partir de um meta modelo já existente. Esse editor será depois sujeito a testes por parte de um grupo de utilizadores que tenham conhecimento prévio do método e que já tenham utilizado outra ferramenta para criar modelos KAOS.

1.4 Principais contribuições previstas

Na sequência deste trabalho foi criado um meta modelo do método com especificações mais rigorosas e com a proposta de um novo conceito (Compartimento) para melhorar problemas existentes, tais como escalabilidade de modelos e complexidade visual. Pretende-se com este novo conceito, conseguir melhorar o problema da escalabilidade de modelos feitos com métodos EROO, pois a sua capacidade de colapsação facilita a leitura dos mesmos através da possibilidade de omitir ou apresentar porções do modelo à vontade do utilizador. Portanto, o trabalho resultante da elaboração desta dissertação vai ser derivar uma ferramenta, à qual foi dado o nome ***modularKAOS***, que verifica a correcção sintáctica dos modelos e vence a sua complexidade visual, além de uma LED que favorece a implementação de ferramentas mais rigorosas da abordagem.

1.5 Organização do documento

Este documento está organizado do seguinte modo:

- Capítulo 1: o presente capítulo, onde é feita uma contextualização do problema, qual o problema encontrado e a solução proposta;
- Capítulo 2: é feita uma breve apresentação da área de Engenharia de Requisitos Orientada a Objectivos (EROO), seguida de um pequeno resumo do método KAOS e as características mais relevantes;
- Capítulo 3: neste capítulo é apresentado um pequeno sumário da área de Modelação Específica do Domínio (MED) e focados alguns pontos relevantes no âmbito da dissertação;
- Capítulo 4: neste capítulo são descritos os passos tomados para a criação da ferramenta;
- Capítulo 5: neste capítulo são descritos quais os caminhos tomados para a validação da ferramenta criada e os respectivos resultados da validação;

- Capítulo 6: é apresentada uma conclusão sobre o trabalho realizado e que outros trabalhos poderão ser realizados a partir do que foi obtido desta dissertação;
- Apêndice A: é apresentado o questionário feito aos utilizadores sobre a ferramenta;
- Apêndice B: é apresentado um pequeno manual de utilizador para futuros utilizadores da ferramenta aprenderem a usar a mesma.

2 . Engenharia de Requisitos Orientada a Objectivos

A **E**ngenharia de **R**equisitos **O**rientada a **O**bjectivos (EROO) pretende utilizar os objectivos (*goals*) para elicitar, elaborar, estruturar, especificar, analisar, negociar, documentar e modificar requisitos [39]. Os objectivos são fins definidos pelas partes interessadas que devem ser atingidos pelo sistema composto e que necessitam da colaboração de agentes para a sua satisfação. Os objectivos podem ser *funcionais* (tarefas que devem ser realizadas pelo sistema) ou *não funcionais* (atributos de qualidade). São importantes na Engenharia de Requisitos porque [39]:

- fornecem um critério preciso para completude suficiente de um requisito, isto é, a especificação de um requisito é completa se o conjunto de objectivos associados puderem ser todos provados a partir da especificação e das propriedades existentes no domínio;
- fornecem um critério preciso para pertinência de um requisito, isto é, um requisito é pertinente em relação a um conjunto de objectivos se a sua especificação justifica a existência de pelo menos um deles;
- uma árvore de refinamento de objectivos apresenta ligações de rastreabilidade entre os objectivos mais abstractos de alto nível e os requisitos mais técnicos de baixo nível;
- o refinamento ajuda a estruturar documentos complexos de requisitos o que facilita a sua legibilidade;
- os refinamentos alternativos fornecem o nível de abstracção adequado para a validação de escolhas a exercer ou sugestão de possíveis alternativas;
- ajudam à detecção de conflitos e respectiva solução;
- os objectivos representam informação mais estável que a informação representada pelos requisitos;
- ajudam à identificação de requisitos para satisfação de objectivos.

Em resumo, os requisitos "implementam" os objectivos da mesma maneira que os programas implementam especificações do desenho de um sistema.

2.1 KAOS

KAOS é um acrónimo para ***K**nowledge **A**cquisition in Aut**O**mated **S**pecification* ou para ***K**eep **A**ll **O**bjectives **S**atisfied*. É uma metodologia para EROO resultante da cooperação entre a Universidade de Oregon (Estados Unidos) e a Universidade de Louvain (Bélgica) em 1990 [20]. Actualmente o método continua a ser desenvolvido e melhorado na Universidade de Louvain por uma equipa liderada pelo Professor Axel van Lamsweerde.

Este método contém um modelo conceptual para adquirir e estruturar requisitos com uma linguagem de aquisição associada e um método de elaboração de modelos de requisitos baseados na linguagem. Com o KAOS é possível definir os objectivos em diferentes níveis de abstracção, desde objectivos de alto nível relacionados com objectivos organizacionais, estratégicos e especificados difusamente ("servir mais passageiros"no caso de um sistema de controlo de comboios) até objectivos de mais baixo nível com especificações mais técnicas, mais detalhadas e relacionadas com o desenho do sistema ("enviar comando de aceleração a cada 3 segundos").

O método de obtenção de requisitos é basicamente constituído pelas seguintes etapas:

- **Etapa de Elaboração de Objectivos** (*goal elaboration step*): elaborar um grafo de refinamento de objectivos a partir de uma descrição preliminar dos mesmos, procurando por palavras chave ou fazendo perguntas de *Como* e *Porquê*;

- **Etapa de Modelação de Objectos** (*object modelling step*): derivar classes, associações e atributos conceptuais a partir da especificação do objectivo;

- **Etapa de Modelação de Agentes** (*agent modelling step*): identificar agentes e as suas capacidades de monitorização/controlo e explorar atribuições alternativas de objectivos a agentes;

- **Etapa de Operacionalização** (*operationalization step*): são identificadas operações e respectivas pré e pós-condições e condições de trigger para garantir a satisfação dos objectivos.

Idealmente o processo segue esta ordem mas na prática os passos podem intercalar-se [42].

A abordagem à aquisição de requisitos baseada no KAOS compreende três níveis:

- meta-nível, onde são definidos os meta-conceitos, as meta-relações e as meta-restrições que se referem a conceitos que devem ser adquiridos para a elicitação de requisitos (ex.: Agente, Requisito, Expectativa);

- nível do domínio, onde estão os conceitos específicos do domínio da aplicação (ex.: Tripulantes da Ambulância, Chamada Urgente);

- nível de instância, onde são declaradas instâncias específicas dos conceitos do nível de domínio.

A linguagem de especificação tem dois níveis: (i) um nível exterior para declarar os conceitos do nível do domínio com uma estrutura entidade-relação; (ii) um nível interior com asserções que caracterizam estes conceitos baseado em lógica temporal .

2.1.1 Conceitos

Esta metodologia compreende os seguintes conceitos [20, 32, 29]:

- **Objecto** (*Object*): coisas de interesse no domínio do sistema que podem sofrer mudanças de estado e que são referidas pelos requisitos. São descritos através de asserções e invariantes e classificados, de acordo com a passividade e dependência, em:
  - Entidade (*entity*): objecto passivo e independente;
  - Agente (*agent*): objecto activo e independente;
  - Associação ou Relação (*association*): objecto passivo e dependente;
  - Evento (*event*): objecto instantâneo, isto é, as instâncias deste objecto apenas existem num estado.

  Entende-se por activo/passivo um objecto que realiza/não realiza operações e dependente/independente um objecto que depende/não depende de outros objectos para sua definição.
  Exemplos de objectos para um problema de organização de encontros seriam: **Entidade**: Encontro, **Agente**: *Participante*, **Associação**: *Desejado* (liga *Participante* e *Encontro*), **Evento**: *PedidoDeEncontro*.

- **Agente** (*Agent*): é um objecto activo que possui comportamento, que faz parte do sistema composto[1] e que realiza um papel específico para satisfação do objectivo. Podem ser, por exemplo, humanos ou dispositivos físicos. Como é uma especialização de Objecto então herda os atributos que o caracterizam como o Nome e a Definição Informal. Um agente pode ser (i) **agente do sistema**, ou seja, pedaços de software do sistema a conceber; (ii) **agente do ambiente**, isto é, um humano ou algo que já existe previamente no domínio. Sendo objectos activos significa que podem realizar operações.
  Além do exemplo dado no ponto anterior (sobre os *Objectos*), podem ser considerados agentes para o mesmo problema do escalonamento de encontros, o *Escalonador* (Agente do Sistema) e o *Iniciante da Reunião* (Agente do Ambiente).

- **Conflito** (*Conflict*) [41]: diz-se que existe um conflito quando dois ou mais objectivos não podem ser satisfeitos em simultâneo dentro de uma determinada condição fronteira, tornando-se logicamente inconsistentes no domínio considerado (ex.: desempenho vs segurança). Para este trabalho considerou-se que a relação de conflito aplica-se apenas entre requisitos não funcionais.
  Um exemplo de um conflito, para um sistema de controlo de elevadores seria, para criar um sistema barato, os requisitos não-funcionais *Custo* e *Robustez* são difíceis de realizar em conjunto (ver figura 2.6).

[1] Conjunto formado pelo sistema a implementar e pelo ambiente que o rodeia.

- **Condição fronteira** (*Boundary Condition*): descreve inconsistências no domínio considerado. Este conceito está relacionado com a noção de conflito, quando este conceito é aplicado a requisitos funcionais. Quando a condição é verdadeira no domínio considerado então existe um conflito entre dois ou mais objectivos relacionados.

- **Obstáculo** (*Obstacle*): quando o comportamento inesperado de um agente impede a satisfação de um objectivo diz-se que existe um obstáculo, ou seja, enquanto um objectivo captura o comportamento desejado pelo sistema os obstáculos capturam comportamentos indesejados [38]. [29] apresenta propostas de detecção e resolução de obstáculos.
  Um exemplo de obstáculo, para o mesmo problema do sistema de elevadores, seria não ser possível ler a direcção do elevador pelos mais variados motivos (ver figura 2.5).

- **Propriedades do Domínio** (*Domain Properties*): propriedades relevantes no domínio da aplicação que são naturalmente verdade sobre o sistema composto. São declaradas como invariantes do domínio ligadas ao objecto no modelo de objectos [29]. Dividem-se em :

  - **Hipóteses do Domínio** (*Domain Hypothesis*): propriedades do domínio do objecto que se espera que mantenham (ex.: "as linhas do comboio estão em boas condições");
  - **Invariantes do Domínio** (*Domain Invariants*): propriedades que são sempre mantidas em todos os estados do objecto (ex.: "as portas do elevador ou estão abertas ou estão fechadas").

- **Acção** (*Action*): as aplicações de uma acção definem transições de estado. Os atributos deste conceito são: (i) *nome* (nome da acção), (ii) *definição informal* (descrição em linguagem natural da acção), (iii) *pré-condição* (condição necessária mais fraca do estado inicial para aplicação da acção), (iv) *condição de gatilho (trigger)* (condição suficiente mais fraca no estado inicial para o despoletar da acção), (v) *pós-condição* (valor que é a condição mais forte do estado final para descrever o efeito da aplicação da acção).

- **Restrição** (*Constraint*): as restrições são formuladas em relação a objectos e acções disponibilizadas a um agente no sistema, isto é, pode ser estabelecida através de transições de estado apropriadas sobre o controlo de agentes. Uma restrição é operacional pois pode ser realizada por aplicação de acções por parte dos agentes. Dividem-se em (i) **Restrições Rígidas** (*Hard Constraints*) que nunca podem ser violadas; (ii) **Restrições Suaves** (*Soft Constraints*) que podem ser temporariamente violadas.

- **Operação** (*Operation*): Uma operação é um requisito cujo agente responsável, através de ligações de operacionalização, realiza serviços funcionais [30]. Além do nome, contém os seguintes atributos:

  - *pré-condição do domínio*: caracteriza os estados antes da aplicação da operação;

- *pós-condição do domínio*: define a relação dos estados antes e depois da aplicação da operação;
- *pré-condição requerida*: define os estados em que é permitida a aplicação da operação;
- *pós-condição requerida*: define condições adicionais que devem ser satisfeitas após a aplicação da operação;
- *condição de gatilho* (*trigger*): define os estados em que é obrigatória a aplicação da operação se a pré-condição do domínio é verdadeira.

Como exemplo de uma operação, para o problema do sistema de elevadores, é apresentada na figura 2.10 a operação *Realizar Escalonamento*.

- **Objectivo** (*Goal*): fim a atingir pelo sistema composto normalmente com a cooperação de agentes. Para uma descrição mais detalhada deste conceito ver secção 2.1.1.1.

2.1.1.1 Objectivo

Cada objectivo tem um nome, uma descrição informal em linguagem natural, onde são descritas as condições para satisfação do mesmo e uma descrição formal opcional, que contém fórmulas em lógica temporal que o descrevem. Intuitivamente, a descrição formal e a descrição informal devem-se referir às mesmas propriedades.

◊ (some time in the future)	♦ (some time in the past)
❑ (always in the future)	■ (always in the past)
W (always in the future *unless*)	B (always in the past *back to*)
U (always in the future *until*)	S (always in the past *since*)

Figura 2.1 Operadores de lógica temporal

No caso do KAOS, os objectivos são identificados através de entrevistas às partes interessadas no sistema, análises de documentos recebidos, refinamento e abstracção através de perguntas "Porquê"e "Como"sobre objectivos ou requisitos já existentes (Porquê (*Why*): porque razão este objectivo existe no modelo; Como (*How*): como o objectivo pode ser satisfeito) e por resolução de conflitos e obstáculos.

Os objectivos são classificados de acordo com o seu padrão e a sua categoria.

O padrão de um objectivo baseia-se no seu comportamento temporal e permite a descrição do comportamento sem escrever especificações formais. Pode ser também usado para guiar a especificação da definição formal do objectivo. São declarados colocando o nome do padrão precedendo o nome do objectivo. A linguagem KAOS considera quatro padrões:

- Alcançar (*Achieve*): alguma acção irá ocorrer no futuro;
- Cessar (*Cease*): alguma acção irá terminar no futuro;
- Manter (*Maintain*): requer que uma dada propriedade se mantenha sempre;
- Evitar (*Avoid*): requer que uma determinada acção nunca aconteça.

Abaixo é apresentado um fragmento correspondente à especificação de um objectivo, mais concretamente um objectivo com o padrão Alcançar [29]:

Objectivo: Alcançar[EncontroConvenienteRealizado]
Definição cada encontro pedido é eventualmente mantido com a presença de todos os participantes pretendidos.
DefFormal $\forall$ m: Encontro:
m.Pedido
$\Rightarrow \Diamond$
m.Mantém $\wedge$ ($\forall$ p:Participante): Intencionado(p,m) $\rightarrow$ Participa(p,m)

Os padrões têm impacto no conjunto de possíveis comportamentos do sistema: os objectivos Alcançar e Cessar geram comportamento, os objectivos Manter e Evitar restringem comportamento. Nesta dissertação não vai ser dada ênfase à parte formal do KAOS.

As categorias dos objectivos fornecem uma classificação mais aprofundada que pode ser usada como guia para aquisição, definição e refinamento de objectivos e é baseada no serviço fornecido aos agentes (funcionais) ou atributos de qualidade do serviço (não-funcionais). Existe uma primeira categorização que divide os objectivos em:

- Objectivos do Sistema (*System Goals*): objectivos específicos do domínio que devem ser atingidos pelo sistema composto;
- Objectivos Privados (*Private Goals*): objectivos específicos dos agentes que podem ser atingidos pelo sistema composto.

Os Objectivos do Sistema podem ser especializados em várias categorias. As categorias consideradas incluem:

- Objectivos de Satisfação (*Satisfaction Goals*): objectivos Alcançar com a preocupação de satisfazer os desejos dos agentes. Os estímulos representam pedidos de serviço, as respostas indicam que os serviços pedidos foram fornecidos;
- Objectivos de Integridade (*Safety Goals*): objectivos Manter que têm como preocupação evitar estados que ponham em perigo a estabilidade do sistema;
- Objectivos de Segurança (*Security Goals*): objectivos Manter que evitam ameaças ao sistema;

- Objectivos de Informação (*Information Goals*): objectivos Alcançar com a finalidade de informar o agente sobre estados no ambiente;
- Objectivos de Precisão (*Accuracy Goals*): objectivos Manter com a preocupação de manter a consistência das informações passadas ao agente sobre o ambiente;
- Objectivos de Robustez (*Robustness Goals*): são importantes na recuperação de falhas.

Muitas vezes é difícil fazer um refinamento correcto dos objectivos: as decomposições são por vezes incompletas e às vezes inconsistentes.
Os padrões de refinamento são usados porque:

- permitem a ocultação da especificação formal ao engenheiro de requisitos;
- permitem o reconhecimento de requisitos e a detecção de refinamentos incompletos;
- tornam as alternativas explícitas.

Os objectivos podem ser refinados em várias combinações alternativas de sub-objectivos, que se ligam ao objectivo-pai através de ligações E ou OU. Um objectivo E-refinado é um objectivo que fica satisfeito se os seus filhos também forem satisfeitos. Um objectivo OU-refinado é um objectivo que precisa de apenas um filho satisfeito para ficar satisfeito.

Existem tácticas de refinamento e padrões de refinamento formal de objectivos. Os padrões de refinamento formal são refinamentos genéricos de objectivos formulados de maneira abstracta. Estes padrões são classificados de acordo com as tácticas de refinamento. Três tácticas importantes são a táctica de decomposição orientada a marcos, a táctica de decomposição orientada a casos e a táctica de decomposição orientada a agentes.

Um marco [40] é uma etapa a atingir no caminho para a concretização de uma actividade. Então a táctica de refinamento orientado a marcos consiste em refinar objectivos introduzindo marcos intermédios para atingir um estado que satisfaça o objectivo final (ver figura 2.2).

Na táctica de refinamento orientado a casos [40], o objectivo-pai é decomposto em várias

Figura 2.2 Decomposição por marcos

situações distintas (casos) que apresentam as circunstâncias necessárias para a satisfação do

Figura 2.3 Decomposição por casos

mesmo (ver figura 2.3).

A táctica de decomposição orientada a agentes sugere dividir um grupo de agentes responsáveis por um objectivo pai, em subgrupos que ficam responsáveis por sub-objectivos do mesmo. Tal como é descrito em [29], existem tácticas de elaboração de especificação cuja aplicação é orientada pela necessidade de resolver objectivos irrealizáveis. A aplicação destas tácticas produz novos modelos de objectivos, modelos de agentes e modelos de objectos.

2.1.2 Modelos do KAOS

O KAOS possui quatro modelos [20], que são descritos nos próximos quatro pontos. Os modelos contemplados no âmbito desta dissertação são os três primeiros, isto é, Modelo de Objectivos, Modelo de Responsabilidades e Modelo de Objectos.

- Modelo de Objectivos (*Goal Model*), que representa os requisitos operacionais, mostra os objectivos do sistema e as relações entre eles. A completude de um Modelo de Objectivos assenta em três pontos: (i) qualidade da fase de definição de requisitos, (ii) reuniões de validação onde as partes interessadas revêm os diagramas de objectivos consolidados, (iii) técnicas de refinamento que garantem completude e consistência. Este modelo possui dois critérios de completude:

  - **Critério de Completude 1**: Um Modelo de Objectivos está completo relativamente à relação de refinamento se e só se todas as folhas do modelo são requisitos, expectativas ou propriedades do domínio.
  - **Critério de Completude 2**: Um Modelo de Objectivos está completo relativamente à relação de responsabilidade se e só se todos os requisitos estão sob a responsabilidade, implícita ou explícita, de um único agente.

  A figura 2.4 apresenta um refinamento E do objectivo de alto nível "Serviço Pedido". Para este objectivo ser satisfeito, os sub-objectivos "Interface do utilizador fornecida", "Interface usada para pedir serviços"e "Utilizadores informados do estado do pedido"devem ser *todos* satisfeitos. Nesta figura são também apresentados agentes do ambiente ("Utilizador") e agentes do sistema ("Sistema de Controlo"e "Designer do Sistema"). "Utilizadores

informados do estado do pedido" é um requisito porque está ligado a um agente do sistema, enquanto "Serviço pedido através da interface" é uma expectativa porque está ligada a um agente do ambiente. Este modelo é completo porque, de acordo com o critério de completude 1 (ver secção 2.1.2), todas as folhas são requisitos ou expectativas e de acordo com o critério de completude 2 (ver mesma secção) os requisitos e expectativas estão atribuídos a um único agente. A figura 2.5 apresenta um obstáculo à satisfação do requisito

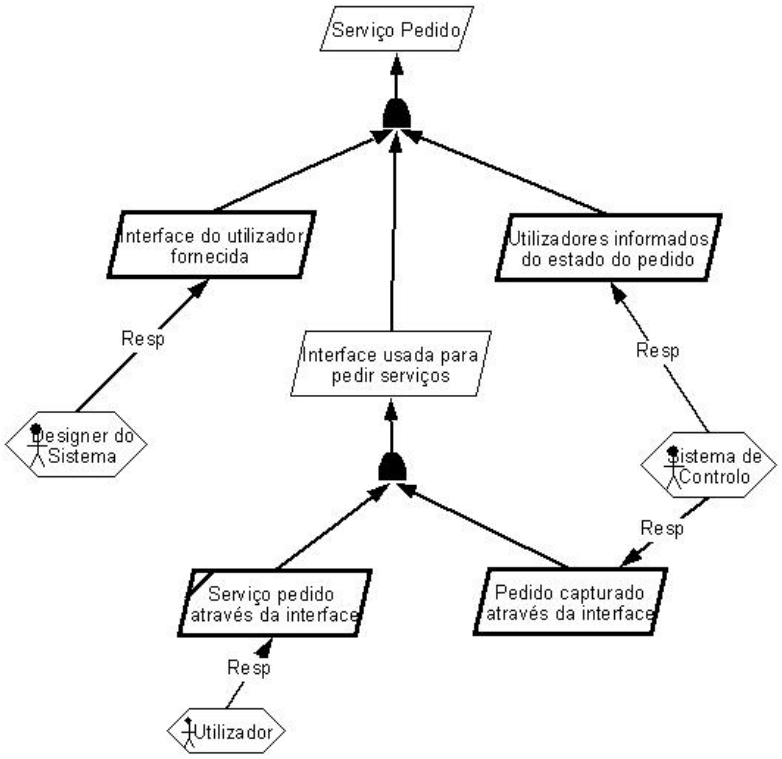

Figura 2.4 Modelo de Objectivos - Serviço Pedido

"Direcção do elevador actualizada alguns segundos antes da próxima paragem". Um obstáculo, tal como um objectivo comum, pode ser refinado. Nesta situação é apresentado um refinamento OU, o que significa que o obstáculo pode acontecer devido a *apenas uma* das situações identificadas. São depois apresentadas soluções para esses obstáculos. A figura 2.6 apresenta uma situação de conflito entre requisitos não funcionais. Para se ter um sistema barato então o mesmo deve ser barato na sua construção (relacionado com o atributo de qualidade "Custo"), barato na sua execução e barato na manutenção (que está relacionado com os atributos de qualidade "Fiabilidade", "Robustez"e "Modifiabilidade"). Contudo existe uma relação de conflito entre os pares de requisitos não funcionais Custo-Robustez e Custo-Fiabilidade o que significa que os dois objectivos não podem ser satisfeitos simultaneamente.

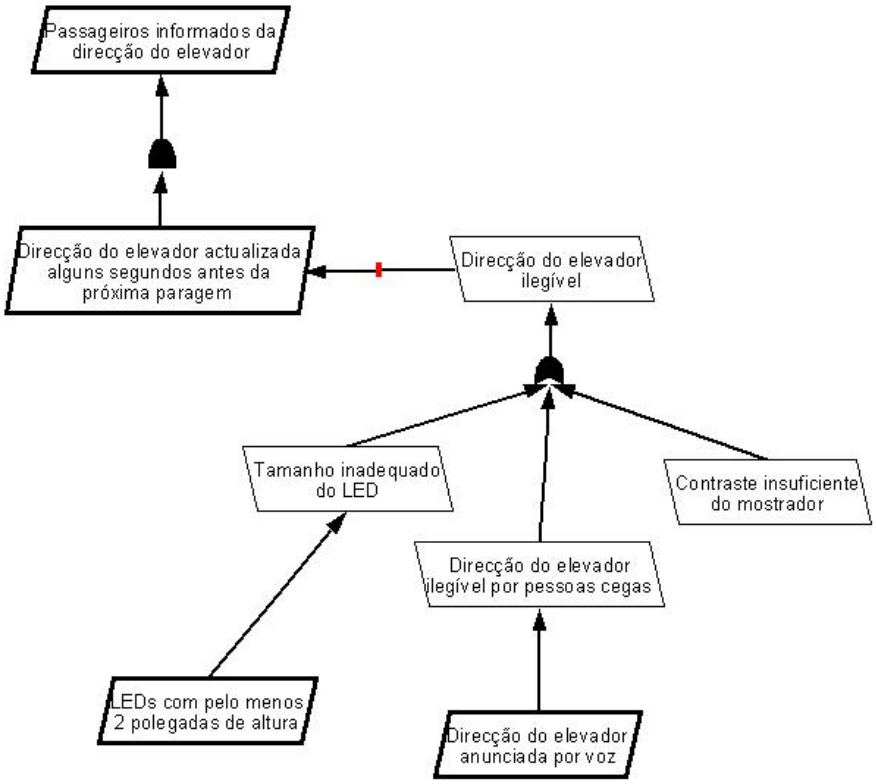

Figura 2.5 Modelo de Objectivos - Passageiros informados da direcção do elevador

- Modelo de Objectos (*Object Model*), que contém toda a informação necessária para produzir o glossário do documento de requisitos, isto é, define e documenta todos os conceitos do domínio que são importantes para a aplicação. Uma parte destes conceitos estão relacionados com interesses das partes interessadas (*stakeholders*) e outra parte refere-se a conceitos necessários para a implementação do sistema. Os objectos são identificados na elaboração do modelo de objectivos. A notação usada é semelhante à utilizada pelos diagramas de classe UML. A figura 2.7 mostra uma ligação de concern entre um objecto (Alarme), um requisito e uma expectativa. Isto significa que a satisfação desses objectivos está relacionada com o conceito representado por esse objecto. A figura 2.8 mostra os componentes de "Sistema de Elevadores", isto é, este objecto é constituído por todos os outros objectos que lhe estão ligados.

- Modelo de Responsabilidades (*Responsibility Model*), que contém todos os diagramas de responsabilidade. Um diagrama de responsabilidade descreve para cada agente, os requisitos e expectativas pelos quais é responsável, ou que lhe foram atribuídos. É construído após todos os requisitos e expectativas terem sido atribuídos a agentes, sendo depois gerado um diagrama para cada agente no sistema. Então este diagrama é derivado do modelo de objectivos.

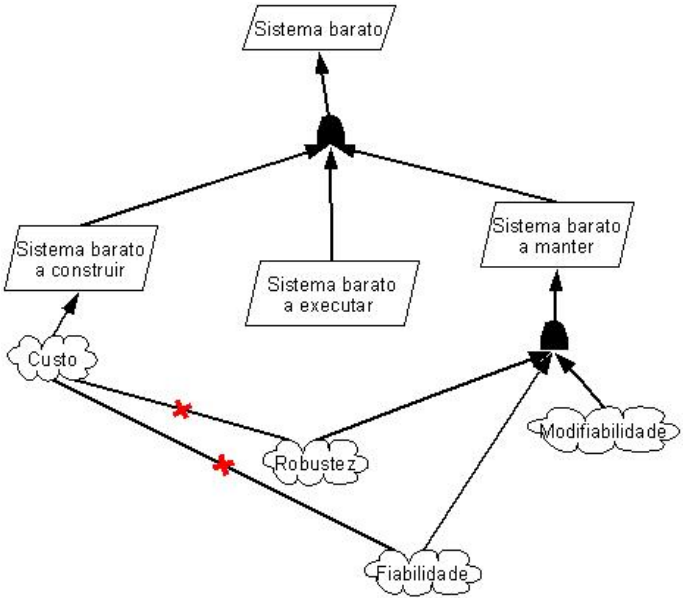

Figura 2.6 Modelo de Objectivos - Sistema Barato

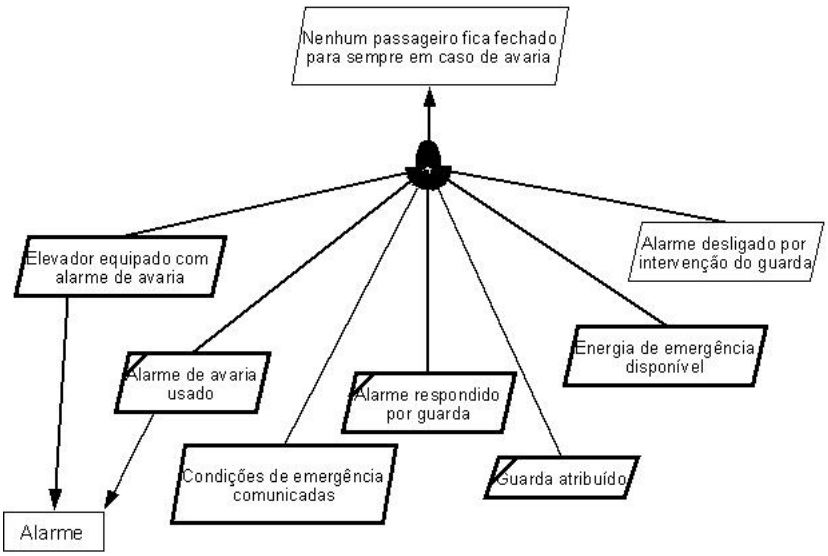

Figura 2.7 Concerns - Sino de Alarme.

A figura 2.9 representa o modelo de responsabilidades do agente "Companhia de Elevadores". Este agente é responsável pela satisfação de todos os objectivos apresentados no modelo.

- Modelo de Operações (*Operation Model*), que descreve todos os comportamentos que os

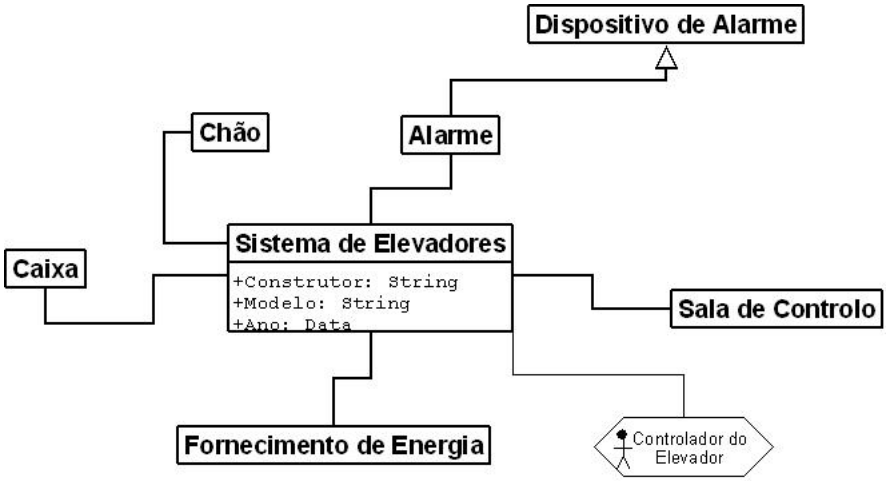

Figura 2.8 Modelo de Objectos do objecto "Sistema de Elevadores".

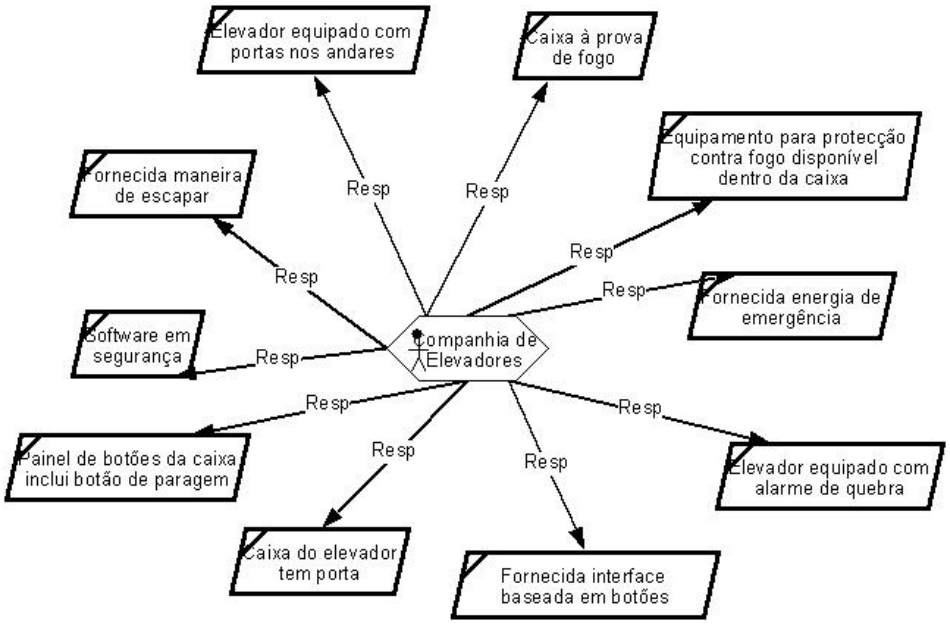

Figura 2.9 Modelo de Responsabilidades - Companhia de Elevadores

agentes precisam realizar para satisfazer os requisitos. As operações podem ser directamente exprimidas pelas partes interessadas durante as entrevistas, ou identificadas durante a análise dos requisitos existentes. Este modelo apresenta três critérios de completude:

– **Critério de Completude 3**: Um modelo de operações para ser completo tem de especificar (i) os agentes que realizam as operações, (ii) os dados de entrada/saída de cada operação.

- **Critério de Completude 4**: Um modelo de operações para ser completo tem de especificar quando as operações vão ser realizadas.
- **Critério de Completude 5**: Todas as operações devem ser justificadas pela existência de um requisito (através do uso de ligações de operacionalização).

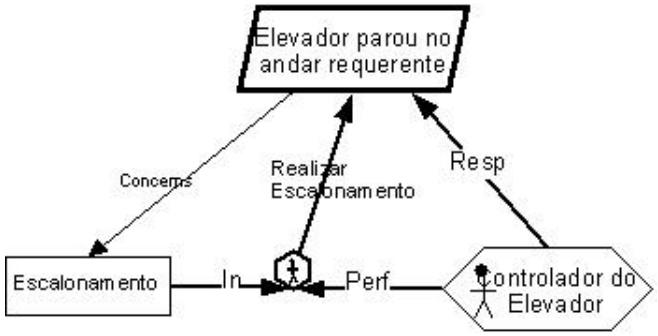

Figura 2.10 Modelo de Operações - Realizar Escalonamento

2.1.3 Vantagens e Desvantagens

Esta metodologia, tal como qualquer outra existente, apresenta vantagens e desvantagens [17]:

- Vantagens
  - Único método da família EROO que dá atenção à prova formal para análise de modelos;
  - Bom para detecção de conceitos e objectivos irrelevantes ou duplicados;
  - Modelos de objectivos e definição de objectivos ajudam à compreensão dos requisitos;
  - A especificação formal ajuda a eliminar a ambiguidade;
  - Dá liberdade à equipa para escolher qualquer método de recolha de dados para construção de conceitos do domínio.
- Desvantagens
  - A lógica temporal não pode ser aplicada a objectivos de alto nível e requisitos não funcionais;
  - Não garante que a equipa escolha a opção correcta;

2.2 Complexidade dos Modelos KAOS

Tal como foi dito na secção 1.2, existem ferramentas para criar modelos KAOS. Foram estudadas duas, a Dia e a Objectiver.

2.2.1 Dia

A Dia [2] é uma ferramenta open-source que pode ser usada para criar não só modelos KAOS, mas também modelos UML e i*, entre outros. Segundo o site, esta ferramenta é inspirada na ferramenta Visio do Windows e é mais orientada para o desenho de diagramas informais sem muita complexidade, por isso, não fornece meios para lidar com a escalabilidade de modelos. Além disso, esta ferramenta permite a criação de ligações ilegais entre conceitos.

2.2.2 Objectiver

A Objectiver [11] é uma ferramenta comercial criada por uma *spin-out* da Universidade de Louvain. Existem quatro edições: *Standard Edition*, *Professional Edition*, *Enterprise Edition* e *Review Edition*. A que foi estudada no decorrer deste trabalho foi a *Professional Edition* que, de acordo com o site, está orientada para projectos de média escala com um simples analista e várias partes interessadas (*stakeholders*). Esta ferramenta, ao contrário da ferramenta anterior, destina-se apenas a criar modelos KAOS e é usada em projectos comerciais com alguma envergadura, como por exemplo, análise de sistemas de segurança para sistemas de transporte aéreo. Contudo esta ferramenta não fornece meios para lidar com modelos grandes o que, em projectos de alguma envergadura, pode torná-los muito complexos visualmente.

2.2.3 Caso de Estudo

As figuras 2.11 e 2.12 referem-se à modelação do sistema **B**ay **A**rea **R**apid **T**ransit (BART) de São Francisco [29, 22, 37]. O principal objectivo é desenvolver um Controlo Avançado de Comboios Automático (*Advanced Automatic Train Control*) que permitirá servir mais passageiros colocando mais comboios com menos tempo de intervalo entre eles. No caso de estudo são focados os aspectos do sistema relacionados com a velocidade e aceleração dos comboios. Estes comboios têm de seguir algumas restrições de segurança tais como, um comboio nunca deve estar tão perto do comboio da frente tal que, no caso de uma paragem repentina do comboio da frente iria ocorrer um choque, andar a uma velocidade inferior à suportada pelo carril, entre outras. Tanto na figura 2.11 como na figura 2.12 o objectivo principal apresenta duas soluções alternativas com alguns obstáculos à sua realização. À medida que o problema ficar mais refinado, é previsível que o diagrama se tornará mais confuso e, ao fim de algum tempo, ilegível e difícil de actualizar. Nos diagramas muito grandes é também complicado analisar as diferentes alternativas individualmente uma vez que são todas visíveis ao mesmo tempo.

As figuras 2.13 e 2.14 também apresentam a modelação do sistema BART, mas com a ferramenta **modularKAOS**. Ambas representam o mesmo, com a diferença que na figura 2.14 existem alguns Compartimentos que estão colapsados para facilitar a leitura do modelo. A ferramenta será descrita mais em pormenor no capítulo 4.

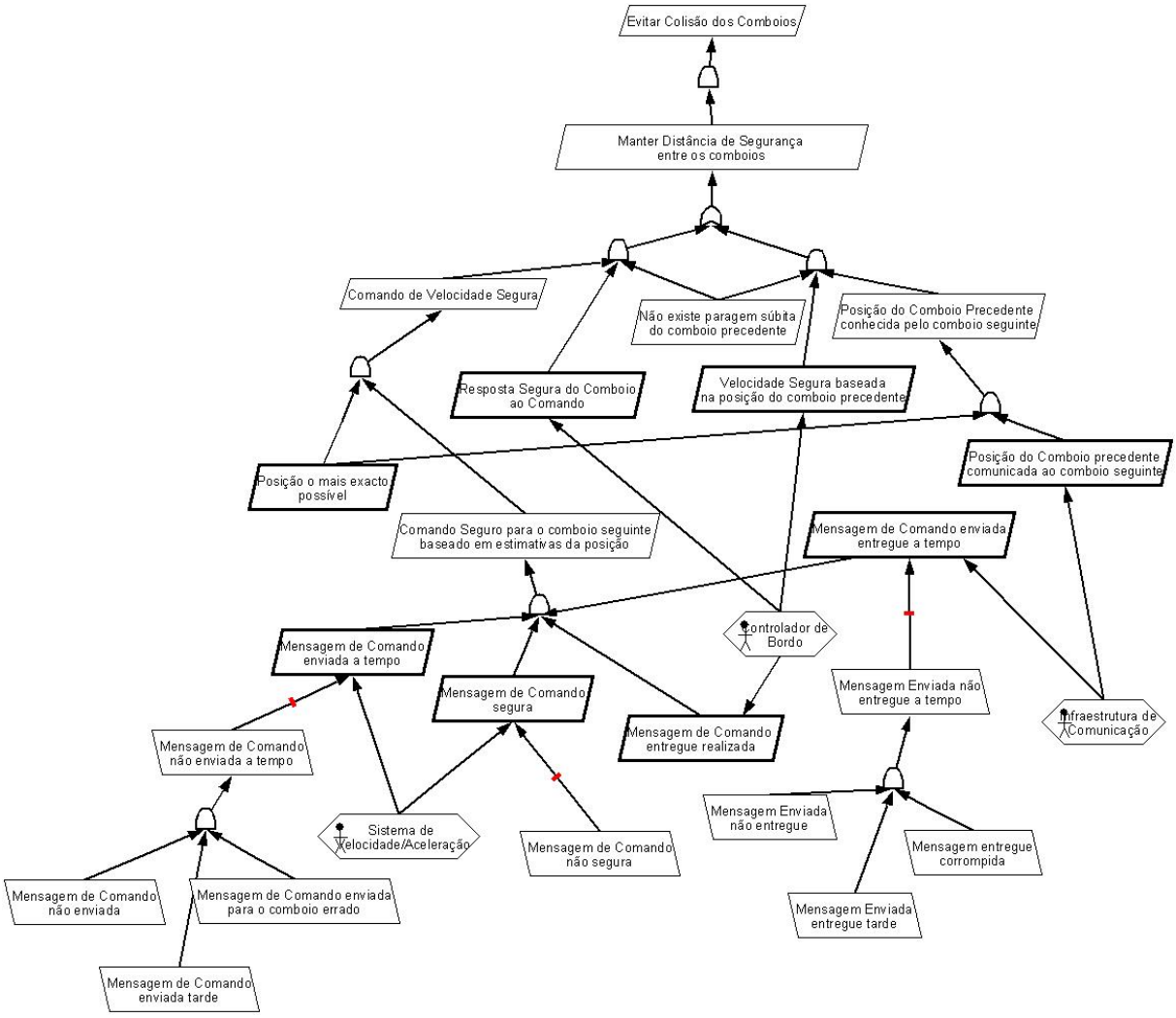

Figura 2.11 Sistema BART modelado com a ferramenta Dia.

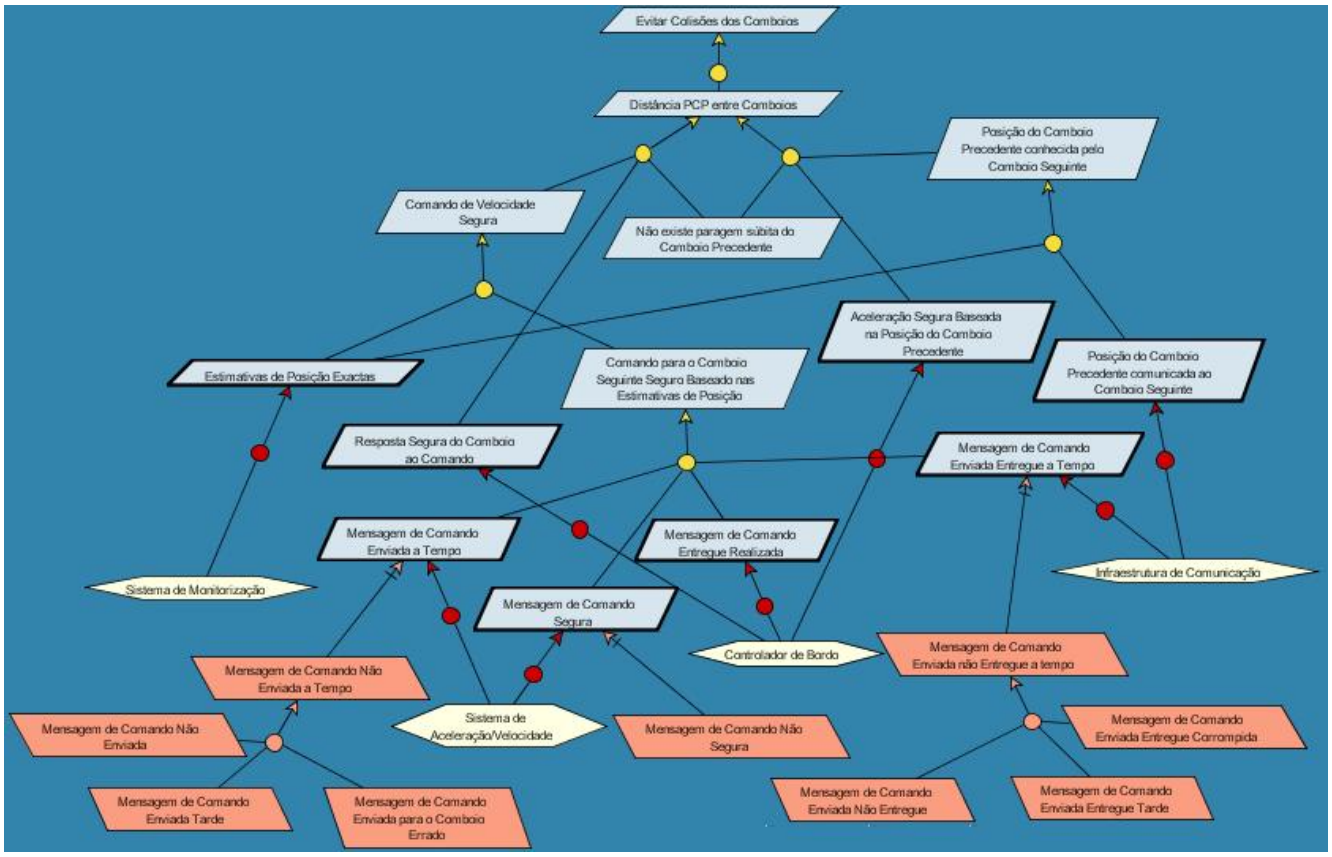

Figura 2.12 Sistema BART modelado com a ferramenta Objectiver.

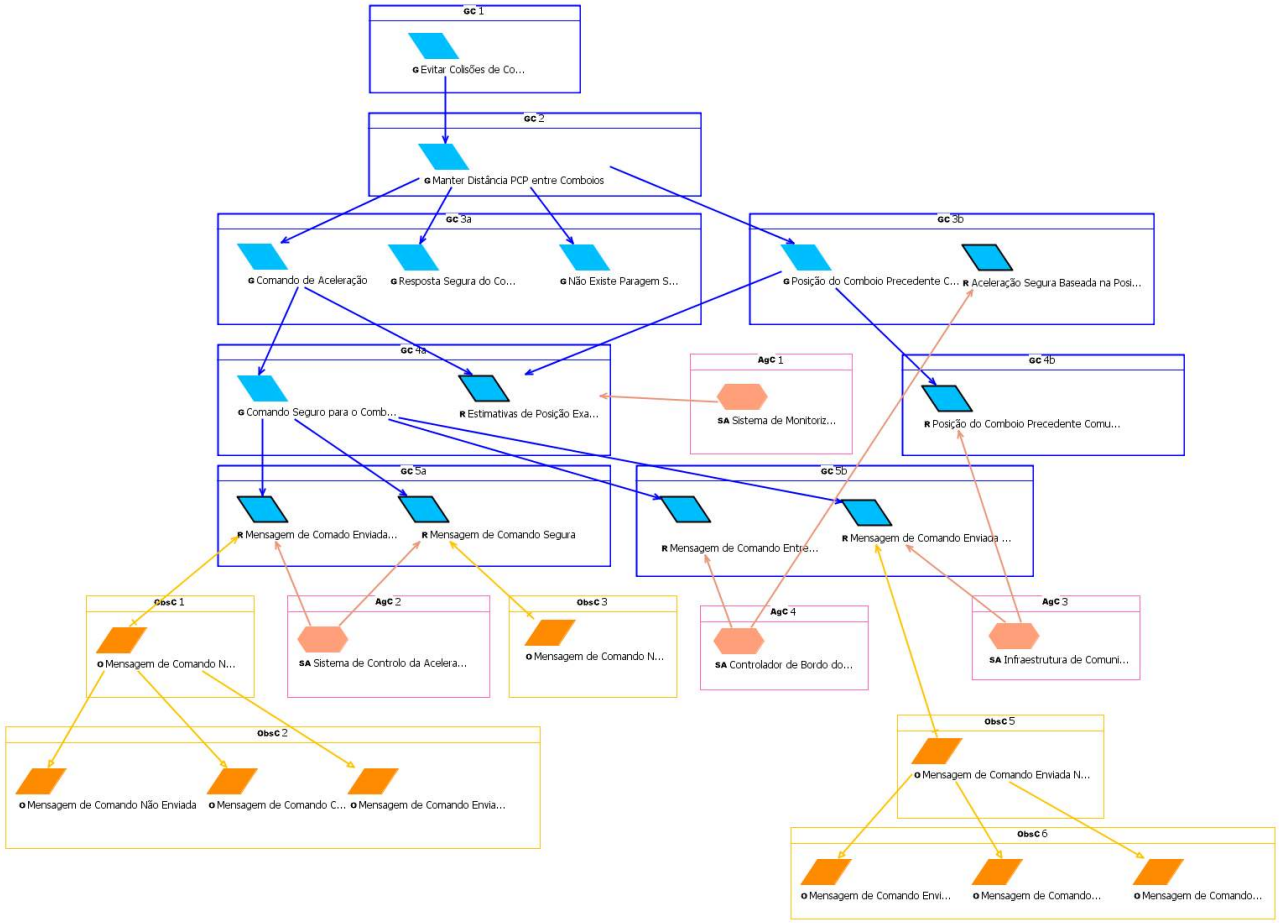

Figura 2.13 Sistema BART modelado com a ferramenta modularKAOS.

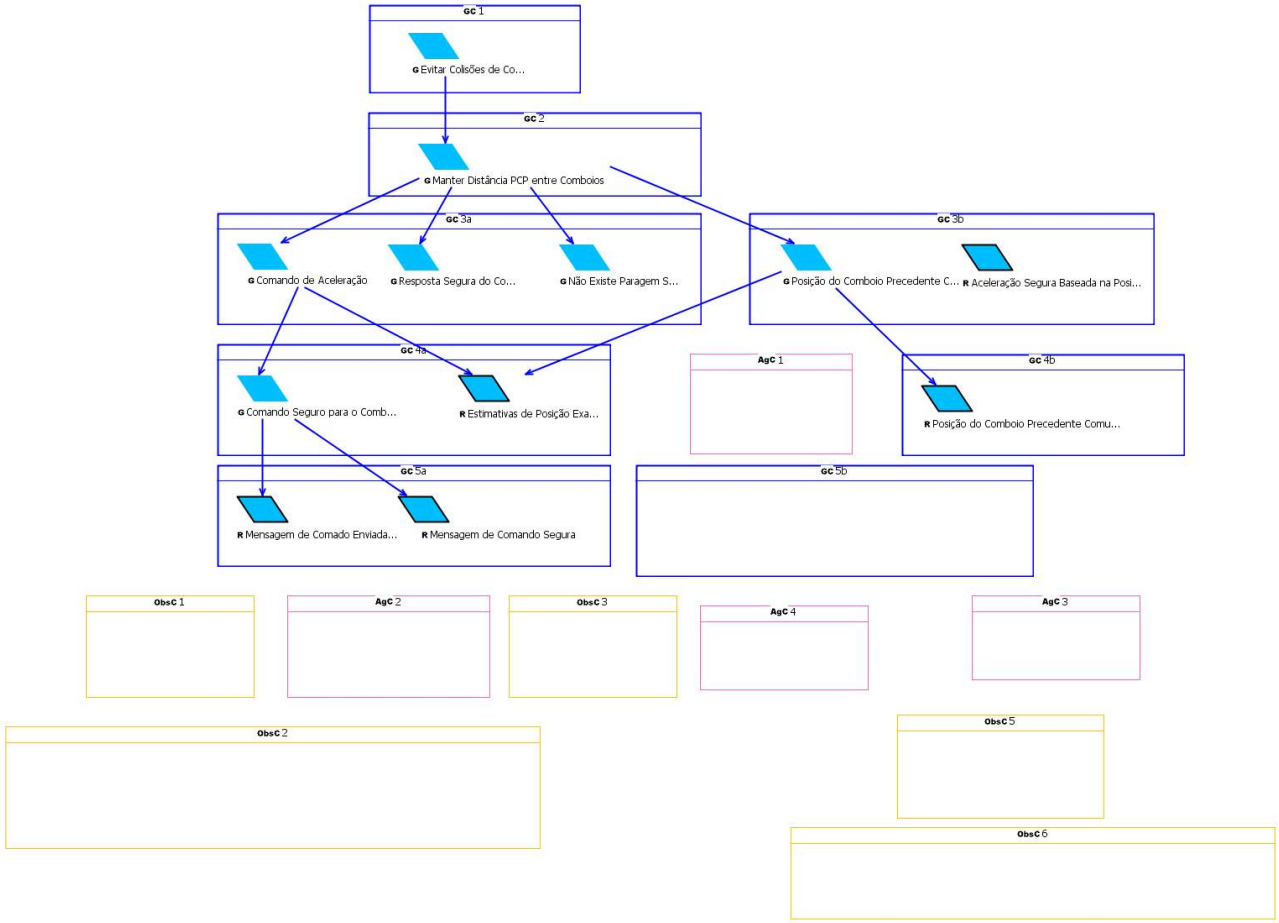

Figura 2.14 Sistema BART com alguns compartimentos colapsados, com a ferramenta modularKAOS.

2.3 Outros métodos de Engenharia de Requisitos Orientada a Objectivos

2.3.1 GBRAM

Goal-**B**ased **R**equirements **A**nalysis **M**ethod (GBRAM) [28] é uma abordagem sistemática para identificação de requisitos e objectivos do sistema que se foca na obtenção, na identificação e na abstracção inicial de objectivos, qualquer que seja a fonte de informação, ao contrário de outros métodos existentes que consideram que já existe documentação sobre os objectivos [27]. Esta metodologia divide-se em duas grandes fases:

- **actividades de análise**, onde os requisitos são identificados (extracção de objectivos e respectivos agentes responsáveis), explorados (exploração da informação existente) e organizados (os objectivos são classificados e organizados de acordo com relações de dependência);
- **actividades de refinamento**, onde os requisitos são refinados, elaborados (identificação de obstáculos e cenários) e operacionalizados (tradução dos objectivos em requisitos operacionais).

Os conceitos-chave do GBRAM são *objectivos*, *partes interessadas* (*stakeholders*) e *agentes* e são identificados na fase de análise e elaborados na fase de refinamento. Estas fases são realizadas com recurso a heurísticas. Consideram-se quatro tipos de heurísticas:

- **heurísticas de identificação**;
- **heurísticas de classificação**;
- **heurísticas de refinamento**;
- **heurísticas de elaboração**.

Esta abordagem faz também uso de perguntas padrão para elicitação e refinamento de objectivos. Com esta metodologia, os elementos estão descritos de forma textual sobre a forma de esquemas (*goal schemas*), isto é, não existe notação gráfica para representar objectivos ou refinamento de objectivos, por exemplo.

2.3.2 i*

i* é um método de modelação orientada a agentes [28]. É constituído por duas grandes componentes: modelo de Dependências Estratégicas (*Strategic Dependency* - SD - figura 2.15) e o modelo Estratégico Racional (*Strategic Rationale* - SR - figura 2.16). O modelo SD mostra as relações de dependência entre actores: captura a intenção dos processos e o que é importante para os seus participantes de forma abstracta. O modelo SR mostra o que está por trás dos processos do sistema: a configuração do processo é descrita com base em recursos, requisitos não

Objectivos	**Restrições**
G54: FAZER declaração online disponível para cada organização	Membro deve estar autorizado a aceder à declaração online
G55: EVITAR compras duplicadas	Membros devem poder saber se a sua organização tinha comprado previamente o produto desejado
G56: FAZER cassetes compradas	Apenas membros do consórcio e participantes do seminário podem comprar cassetes do seminário

Tabela 2.1 Exemplo de um *schema* em GBRAM (retirado de [16]).

funcionais, tarefas, entre outros, apresenta um nível de abstracção mais baixo, permitindo analisar com grande detalhe o processo interno dentro de cada actor. O conceito mais importante nesta metodologia é o de *actor*. Os actores têm propriedades intencionais como objectivos, crenças (*beliefs*), habilidades (*abilities*) e compromissos (*commitments*). Os actores dependem uns dos outros para satisfação dos objectivos, fornecimento de recursos, execução de tarefas que não pode fazer por si. Os actores podem ser humanos ou sistemas com capacidades específicas. Os tipos de dependência entre os actores podem ser classificados de quatro maneiras, dependendo do objecto de dependência (isto é, se é um objectivo, requisito não funcional, tarefa ou recurso).

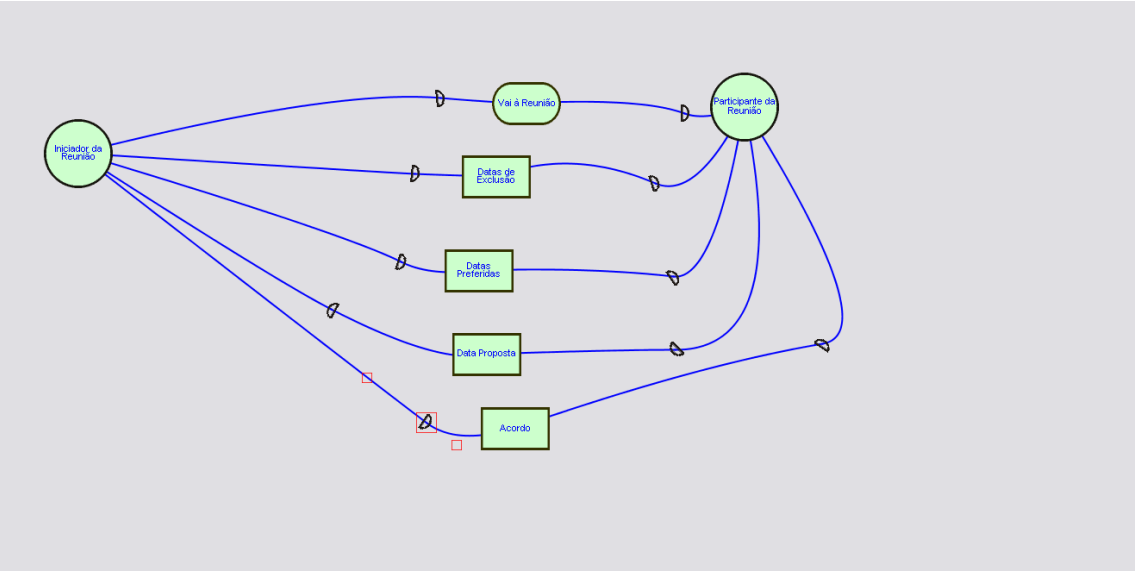

Figura 2.15 Exemplo de um modelo i* - modelo SD (retirado de [12]).

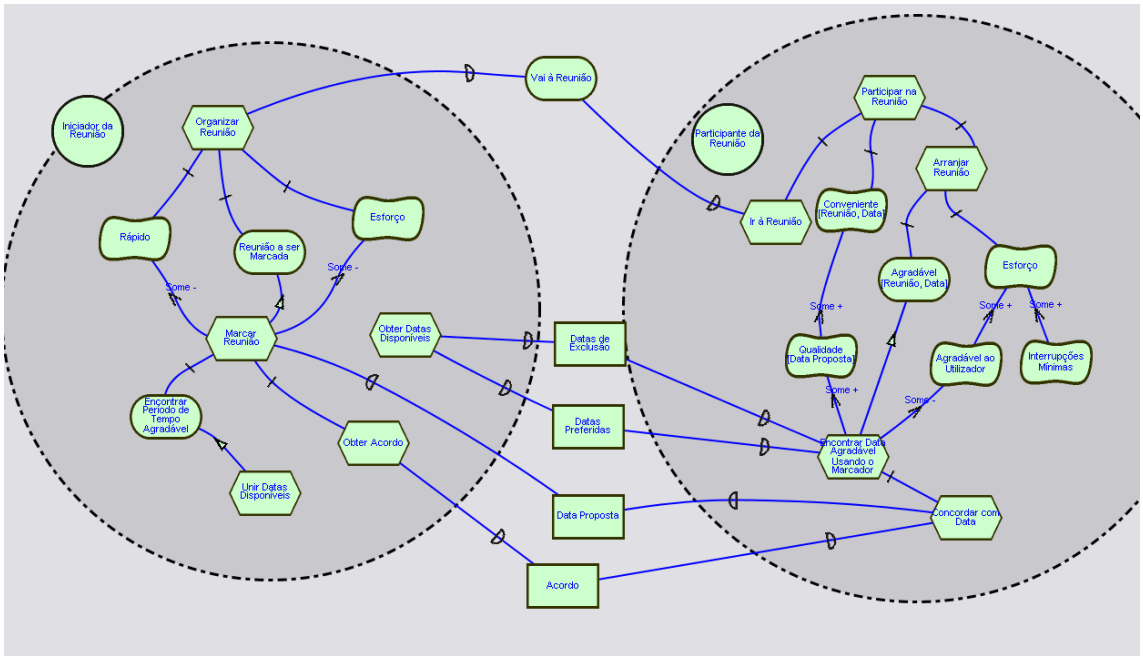

Figura 2.16 Exemplo de um modelo i* - modelo SR (retirado de [12]).

2.3.3 NFR

Non-**F**unctional **R**equirements (NFR) Framework [28] está relacionada com a análise e modelação de requisitos não-funcionais. Os objectivos principais são: capturar os requisitos não funcionais (NFR's) com interesse para o domínio, decompô-los, identificar possíveis operacionalizações, lidar com ambiguidades, prioridades e inter-dependências entre eles, seleccionar operacionalizações, apoiar decisões com o desenho e avaliar o impacto das escolhas. A ideia principal é modelar e refinar os requisitos não-funcionais e apresentar as influências positivas e negativas nos requisitos. Os requisitos não-funcionais podem ser refinados com ligações E e OU e as suas inter-dependências são representadas através de ligações de contribuição positiva (+) e contribuição negativa (-). Esta metodologia é modelada através de um grafo de interdependência de requisitos não-funcionais. Neste grafo estão representados os requisitos não funcionais, os refinamentos, as contribuições e operacionalizações. O refinamento pára quando se atinge um requisito não funcional que esteja suficientemente detalhado, normalmente um requisito não funcional operacional. Com este grafo é possível visualizar as diferentes alternativas e escolher quais as que poderão melhor satisfazer os requisitos não-funcionais de alto nível do sistema.

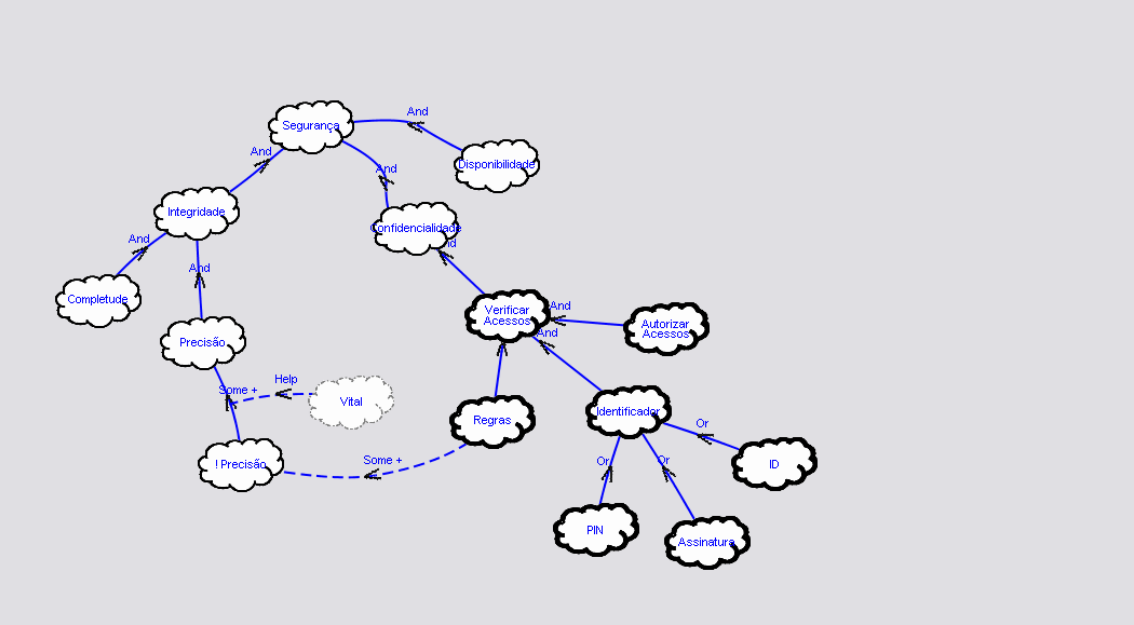

Figura 2.17 Exemplo de um modelo NFR (retirado de [12]).

2.3.4 GRL

Goal-oriented **R**equirement **L**anguage (GRL) [9] é uma linguagem para modelação orientada a objectivos e a agentes e para análise de requisitos, em especial requisitos não-funcionais. Com esta linguagem é possível exprimir vários conceitos que surgem durante o processo de obtenção de requisitos. Existem três categorias de conceitos: (i) **elementos intencionais** (objectivo, tarefa, requisito não funcional, crença e recurso), (ii) **relações intencionais** (*means-ends*, decomposição, contribuição, co-relação, dependência), (iii) **actores** (entidade activa que realiza acções para atingir objectivos). São chamados de intencionais porque são usados nos modelos para responder a questões sobre o porquê do comportamento, a escolha de determinados aspectos estruturais e informativos, alternativas consideradas, critérios usados para as escolhas e porque razão essa alternativa foi escolhida. Com esta modelação, a principal preocupação do analista é explicar porque determinada estrutura ou comportamento foi escolhida ou porque alguma restrição foi inserida. O analista não está interessado nos detalhes operacionais dos processos ou requisitos do sistema. O objectivo é evitar vulnerabilidades expressas pelas partes interessadas utilizando capacidades de outras partes, atribuindo responsabilidades ou introduzindo restrições sobre os comportamentos que devem ser tomados. O GRL ajuda a fazer a correspondência entre os elementos intencionais da linguagem e os elementos não-intencionais dos modelos de cenários.

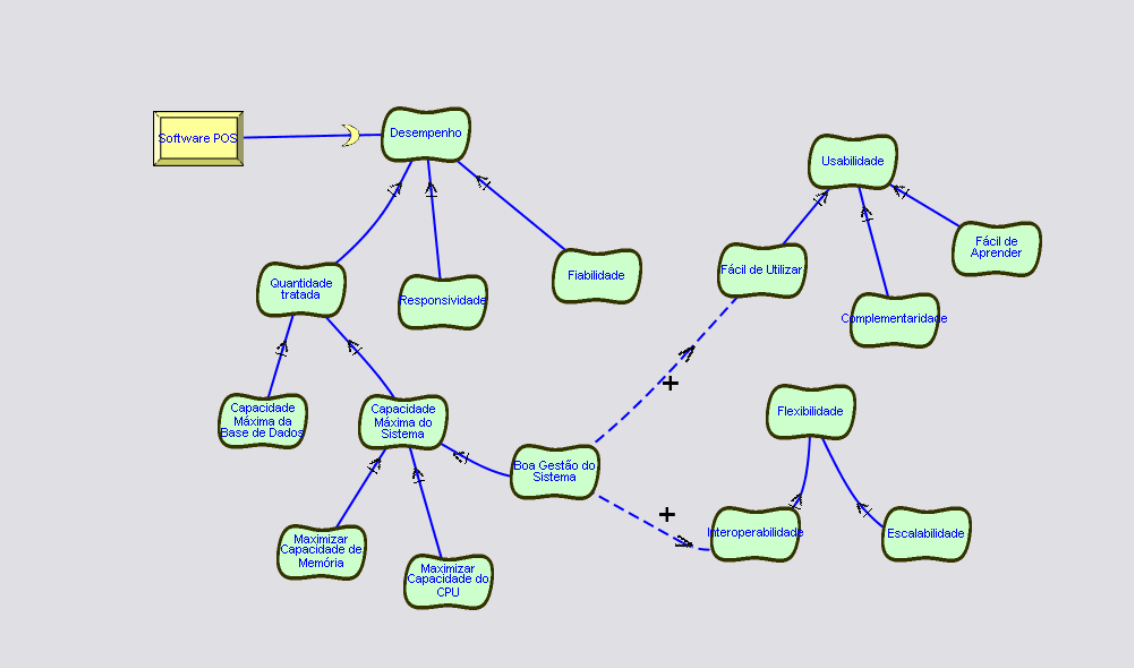

Figura 2.18 Exemplo de um modelo GRL (retirado de [12]).

2.4 Sumário do Capítulo

Neste capítulo foi apresentada a área da Engenharia de Requisitos Orientada a Objectivos (EROO). Foi dado ênfase ao método KAOS, que foi o método escolhido para implementação da ferramenta criada. Deste método foram descritos os conceitos, modelos existentes, vantagens e desvantagens. Foram também mencionadas ferramentas existentes para a criação de modelos KAOS e uma pequena análise das mesmas com um caso de estudo existente. No final foram descritos outros métodos EROO existentes.

O próximo capítulo trata da área da Modelação Específica do Domínio (MED).

3 . Modelação Específica do Domínio

A Modelação Específica do Domínio (**D**omain-**S**pecific **M**odelling - DSM) [21][26] baseia-se na ideia que muitos problemas de desenvolvimento de software podem ser resolvidos desenhando uma Linguagem Específica do Domínio (do inglês *Domain Specific Language*). A **M**odelação **E**specífica do **D**omínio (MED) aumenta o nível de abstração porque usa conceitos do domínio do problema para expressar a solução. Cada solução MED foca-se em pequenos domínios porque um foco mais estreito dá mais hipóteses de automatatização e são mais fáceis de definir. Este método de desenvolvimento de software é um meio de resolver problemas que se aplica quando um problema particular ocorre com frequência. Normalmente as soluções são aplicadas em relação a um produto particular, linhas de produtos ou plataformas.

A Modelação Específica do Domínio tem dois grandes objectivos: (i) aumentar o nível de abstracção através da utilização de conceitos do domínio do problema e (ii) gerar produtos finais numa linguagem de programação escolhida ou outra forma de especificação de alto nível. Apesar de o tempo de implementação de uma MED ser relativamente curto (de algumas semanas a meses), a sua aplicação é adequada quando a previsão de trabalho com o domínio em consideração for grande. Por exemplo, numa linha de produtos, um caso típico de uso de MED é especificar a variação do produto, ou seja, como os produtos são diferentes entre si. Este tipo de modelação também é adequado quando existem especialistas do domínio, não programadores, que podem fazer especificações completas usando a sua própria terminologia e correr geradores para gerar código da aplicação.

Um processo MED apresenta tipicamente as fases apresentadas na figura 3.1.

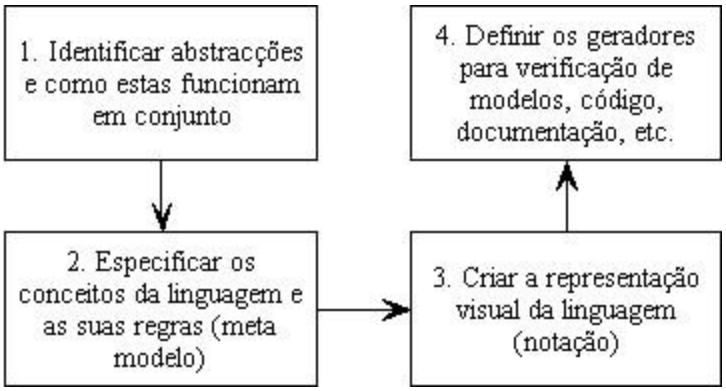

Figura 3.1 Fases num processo MED típico [24].

3.1 Pontos chaves

Os pontos chave da MED são [26]:

- *Tamanho do domínio da aplicação*: o foco pequeno de aplicação significa que a solução não pode ser aplicada em outra situação fora do domínio, o que facilita a automatatização e construção de linguagens eficazes. Um domínio pequeno opera sobre conceitos conhecidos do domínio e tem regras de aplicação. Isto previne erros numa fase inicial da implementação, quando estes são mais baratos de corrigir. O desenho da solução é guiado pelo meta modelo da linguagem fazendo com que não seja possível, por exemplo, fazer ligações ilegais entre elementos. Como se foca em conceitos precisos e conhecidos, os modelos de MED são mais fáceis de ler, recordar, verificar e validar. Em MED, os geradores de código lêem os modelos baseados no meta modelo da linguagem. Um domínio pequeno faz com que os geradores produzam código eficiente, com padrões e estruturas específicas.

- *Alto nível de abstracção*: a MED aumenta o nível de abstracção porque utiliza conceitos do domínio para exprimir a solução. A linguagem de modelação segue as abstracções e semântica do domínio, isto é, os conceitos da linguagem mapeiam os conceitos do domínio. Esta proximidade entre a linguagem e o domínio do problema oferece vários benefícios tais como, maior produtividade, dissimulação da complexidade e melhor qualidade do sistema. Os geradores de código mapeiam os modelos para o domínio da solução, isto é, especificam como a informação é extraída dos modelos e transformada em código. O gerador por si é normalmente invisível aos modeladores e a construção e modificação do mesmo é feita por programadores experientes.

- *Geração de código*: em MED o código é gerado pela aplicação, não sendo necessário escrever código à mão. Este código gerado pode ser ligado com código já existente e compilado sem esforço manual adicional. O código gerado abrange estruturas estáticas e comportamentais. A especificação do problema do domínio pode ser representada na forma textual, diagramas gráficos, tabelas e matrizes. Em MED é possível criar várias linguagens para as diferentes vistas de um modelo. Estas podem ser depois integradas através de linguagens de modelação ou usando linguagens diferentes que são integradas no momento de geração de código.

3.2 Implicações, vantagens e desvantagens

As implicações da utilização de MED são:

- os modelos são usados para gerar código assim como para executar, testar e fazer debug da aplicação ou funcionalidades desenvolvidas;

- não há necessidade de aprender linguagens e semânticas novas;

- as tarefas rotineiras são minimizadas, porque os geradores automatizam tarefas repetitivas;

- é feito menos trabalho de especificação porque é tudo mantido em alto nível;
- é necessário fazer menos testes e acontecem menos erros;
- não é necessário gerar código.

As vantagens da Modelação Específica do Domínio são:

- as **L**inguagens **E**specíficas do **D**omínio (LED) (secção 3.3) permitem que se trabalhe apenas no âmbito do problema levando a que se cometam menos erros do que quando se pretende representar o problema numa Linguagem para Domínios Gerais (*General Purpose Language*);
- ao trabalhar dentro do espaço do problema os modelos são mais facilmente entendidos por pessoas não familiarizadas com a tecnologia de implementação;
- os modelos expressos com a LED podem ser validados ao nível de abstracção do problema, que permite a detecção mais precoce de problemas de entendimento e definição na fase de desenvolvimento;
- os modelos podem simular a solução directamente, fornecendo resposta imediata da adequação do mesmo;
- uma Linguagem Específica do Domínio fornece uma API específica do domínio para manipulação dos modelos, que melhora a produtividade do encarregado do desenvolvimento da solução.

As desvantagens são:

- investimento inicial no desenho e construção da linguagem e integração na solução geral;
- custos de testes, implementação, documentação, treino de pessoal e modificações no processo de desenvolvimento.

3.3 Linguagens Específicas do Domínio

Uma LED (**L**inguagem **E**specífica do **D**omínio) é uma linguagem de programação dedicada a um domínio particular. As LED's fazem parte das linguagens de programação de quarta geração (4GL)[1]. Um exemplo de uma LED é a linguagem SQL, usada para realizar *queries* a bases de dados. As LED's podem ser textuais ou gráficas. Muitas pessoas, normalmente pessoas com experiência de programação, preferem as linguagens textuais para input, porque podem inserir

[1] Linguagens de programação de alto nível com objectivos específicos. Este tipo de linguagens descrevem o que *deve* ser feito em oposição a *como* deve ser feito, característico das 3GL (Linguagem desenhada com o propósito de ser compreendida por humanos. Exemplos deste tipo de linguagens são o ALGOL, C ou Java [14].)[3]

o texto mais rapidamente com o teclado, mas linguagens gráficas para output porque facilita a visualização do resultado de um modo abrangente.

Tal como já foi mencionado as LED's podem ser textuais ou gráficas. Existem três abordagens para implementação de linguagens textuais. Na primeira abordagem pode ser especificada uma gramática (baseada por exemplo em BNF) e depois criar um *parser* para conversão. A implementação realizada deste modo pode ser uma tarefa complicada e sujeita a erros, que deve ser realizada por alguém especializado na actividade. Isto acontece porque a gramática pode ser ambígua ou inconsistente e necessitar de um *look-ahead* grande para decidir o que fazer a seguir. A segunda abordagem é utilizar propriedades de outra linguagem para emular as capacidades de uma linguagem específica do domínio, por exemplo, usar classes e estruturas previamente definidas para criar configurações específicas de objectos e dados para resolução do problema em estudo. A terceira abordagem é usar XML. Desta maneira podem-se definir *tags* que representam o elemento. Esta abordagem tem como vantagem a grande variedade de bibliotecas e ferramentas para processar documentos XML. Pode também ser criado um XML *Schema* que ajuda à validação da correcção do documento editado.

As LED's gráficas têm vários aspectos importantes a ter em conta. Os mais importantes são: (i) *notação* - convenções diagramáticas da linguagem; (ii) *modelo do domínio* - modelo dos conceitos descritos na linguagem; (iii) *geração* - a partir dos modelos criar alguns artefactos como código, data, ficheiros de configuração, outro diagrama ou uma combinação destes; (iv) *serialização* - após criação dos modelos, pretende-se guardá-los, verificá-los com alguma secção de controlo e recarregá-los mais tarde; (v) *integração com ferramentas* - que extensões de ficheiros estão associados com a linguagem, que ícones aparecem na *toolbox*, quais as propriedades dos elementos que são apresentados, que editores utilizar.

3.3.1 Ferramentas de modelação

Existem várias ferramentas que permitem gerar LED's gráficas. Foram consideradas três, cujas descrições são apresentadas de seguida:

- EMF/GMF [4][8]: o **E**clipse **M**odeling **F**ramework (EMF) é uma *framework* para modelação e geração de código para construção de ferramentas e outras aplicações baseada em modelos estruturados. A partir do modelo especificado em XMI[2], são fornecidas ferramentas e suporte em *runtime* para produzir um conjunto de classes Java para o modelo, um conjunto de classes para visualização e edição do modelo e um editor. O EMF é constituído por três partes fundamentais:
  - EMF - o *core* da ferramenta, contém um ficheiro Ecore onde é descrito o meta modelo da linguagem a implementar e uma API para manipulação de objectos EMF.
  - EMF.edit - esta *framework* contém classes usadas para gerir editores baseados em EMF.

[2]Standard OMG para troca de meta dados através de XML.

  - EMF.codegen - aqui é gerado o código para construir o editor da linguagem para o modelo EMF.

  Podem ser inseridas restrições adicionais ao meta modelo criado com esta *framework* utilizando OCL (secção 3.4.2). O Eclipse **G**raphical **M**odeling **F**ramework (GMF) fornece uma componente generativa e uma infraestrutura para desenvolver editores gráficos baseados em EMF e GEF[3]. Estas duas ferramentas estão disponíveis como plugins para o Eclipse.

- GME [7]: Ferramenta desenvolvida pela Universidade de Vanderbilt, nos Estados Unidos da América. O *Generic Modeling Environment* (GME) é uma ferramenta configurável para criar modelos específicos do domínio. A configuração é realizada com recurso a meta modelos que especificam o paradigma de modelação do domínio da aplicação. A linguagem de metamodelação é baseada na notação do diagrama de classes UML e faz uso de restrições OCL. Os meta modelos que especificam o paradigma da modelação são usados para gerar automaticamente o ambiente alvo específico do domínio. O ambiente específico do domínio é usado para construir modelos do domínio que são armazenados na base de dados do modelo ou em formato XML.

- DSL Tools[19]: plugin para a plataforma do Visual Studio da Microsoft. É um conjunto de ferramentas para criar, editar, visualizar e usar dados específicos do domínio para automatizar o processo de desenvolvimento de software. Esta ferramenta está relacionada com o conceito de *software factories*[4]. Com esta ferramenta é possível definir um modelo do domínio com um designer e um gerador de artefactos textuais. Contém também um editor gráfico para definir e editar modelos de domínio. Os dados são guardados num ficheiro XML. O código é gerado a partir da definição do modelo do domínio e da definição do designer. Eventuais restrições que se pretendam adicionar ao meta modelo criado são implementadas em C#.

Para a realização deste trabalho foi escolhido o EMF/GMF (ver capítulo 4).

3.4 Outros conceitos importantes

As próximas subsecções destinam-se a falar de alguns conceitos relacionados com a MED que são importantes para a criação da ferramenta modularKAOS: Modelos Ecore e Linguagem OCL.

[3]Graphical Editing Framework: permite o desenvolvimento de editores gráficos através de um modelo de aplicação existente [5].

[4]É uma linha de produtos de software que une ferramentas de desenvolvimento extensíveis com padrões ou LED's, baseado em receitas para construir tipos específicos de aplicações [13].

3.4.1 Modelo Ecore

Para a criação do meta modelo usado como base para a criação da Linguagem Específica do Domínio que valida os modelos criados, foi usado o modelo Ecore através do plugin EMF do Eclipse.

Um modelo Ecore é uma metalinguagem usada pelo EMF para definir modelos [18]. Os seus padrões são baseados no **M**eta-**O**bject **F**acility versão 1.4 (MOF 1.4) da **O**bject **M**anagement **G**roup (OMG). Permite instanciação de modelos orientados a objectos e é usado como base para geração de código e padrão para modelação de metadados. O modelo estrutural do Ecore permite que se foque apenas em informação essencial e, como é baseado em EMF, contém ferramentas que suportam geração de código e importação/exportação para vários formatos. Os conceitos chave são:

- *EClass*: representa um tipo que pode definir qualquer número de super-tipos, qualquer número de associações e qualquer número de atributos;
- *EAttribute*: representa um atributo de um tipo;
- *EReference*: representa um lado de uma associação e define o tipo referenciado.

Um modelo Ecore pode ser definido dos seguintes modos:

- editor de XML: os ficheiros Ecore são ficheiros XML, por isso é possível defini-los num editor de XML;
- editor de Ecore: a ferramenta EMF fornece um editor de Ecore;
- ferramentas de UML (Rational Rose, EclipseUML): os modelos Rational Rose marcados podem ser convertidos para ficheiros Ecore. O EclipseUML fornece suporte para EMF;
- XML Schema Definition (XSD): um modelo Ecore pode ser definido a partir de um schema;
- interfaces Java: um modelo Ecore pode ser gerado a partir de Java anotado;
- Emfatic: editor de texto que suporta navegação, edição e conversão de modelos Ecore usando uma sintaxe semelhante ao Java [6].

3.4.2 Object Constraint Language

A linguagem OCL é importante no contexto deste trabalho porque ajuda a criar mais restrições, para além das possíveis de criar com o modelo Ecore, de modo a melhorar a consistência e fiabilidade dos modelos.

A **O**bject **C**onstraint **L**anguage (OCL) é uma linguagem textual e precisa para exprimir expressões e restrições em modelos orientados a objectos e outros artefactos de objectos. As

expressões podem ser usadas para: (i) especificar o valor inicial de um atributo, (ii) especificar uma regra de derivação de um atributo, (iii) especificar o corpo de uma operação, (iv) indicar a instância de um diagrama dinâmico, (v) indicar uma condição de um diagrama dinâmico, (vi) indicar os parâmetros actuais num diagrama dinâmico. As restrições podem ser de quatro tipos: (i) *invariante* que indica que uma condição deve ser sempre verdadeira em todas as instâncias da classe, tipo ou interface, (ii) *pré-condição* é uma restrição que deve ser verdadeira antes da aplicação de uma operação, (iii) *pós-condição* é uma condição que deve ser verdadeira no momento em que a operação acabou a sua execução, (iv) *guarda* é uma restrição que deve ser verdadeira antes que uma transição de estado dispare [10]. O OCL usa uma sintaxe não simbólica e contém um número restrito de conceitos. Um dos aspectos mais importantes da linguagem é que faz parte do UML que é uma linguagem de modelação de acordo com os standards da OMG [23]. De acordo com [43], o OCL tem os seguintes requisitos:

- poder expressar informação extra (necessária) dos modelos e outros artefactos usados em desenvolvimento orientado a objectos;
- ser uma linguagem precisa e não ambígua que possa ser facilmente lida e escrita por qualquer praticante de tecnologias de objectos e os seus clientes. Isto significa que a linguagem deve ser entendida por pessoas que não sejam da área da Matemática ou da Informática;
- ser uma linguagem declarativa. A sua expressão não pode ter efeitos colaterais, isto é, o estado do sistema não pode mudar por causa da restrição OCL;
- ser uma linguagem tipificada.

Tem contudo algumas desvantagens:

- esta linguagem está mais relacionada com uma linguagem de implementação do que uma linguagem conceptual, porque usa operações nas suas restrições e o seu sistema de tipos está mais próximo do das linguagens orientadas a objectos;
- as expressões em OCL são por vezes muito verbosas;
- as expressões são por vezes difíceis de ler;
- a linguagem não pode coexistir sozinha, precisa sempre de um diagrama UML onde seja aplicada.

3.5 Sumário do Capítulo

Neste capítulo foi descrita a Modelação Específica do Domínio (MED), fazendo referência aos seus principais objectivos, pontos chave, vantagens e desvantagens. Foi também feita referência

às Linguagens Específicas do Domínio (LEDs) bem como ferramentas existentes para a sua criação. De seguida foram mencionados alguns conceitos adicionais importantes para o âmbito do trabalho feito.

No próximo capítulo vai ser descrito o trabalho realizado nesta dissertação, trabalho esse baseado no conhecimento obtido a partir do estudo realizado acerca de conceitos relevantes, referidos neste capítulo e no capítulo 2.

4. Implementação da Ferramenta

Para desenhar o editor foi escolhida a *framework* Eclipse com os *plugins* EMF/GMF. Esta *framework* foi escolhida porque, dentro das ponderadas (ver secção 3.3.1), foi considerada a mais adequada para o trabalho a realizar pois em termos de definição de interface visual e em termos de possibilidades de modelação, oferece os melhores métodos, tendo um vasto número de opções disponíveis para se fazer uma modelação correcta e completa do domínio em análise; geração de código, pois ao gerar o modelo no editor é também criado um ficheiro XML com a descrição do modelo criado; além disso esta *framework* é *freeware* e já existe há algum tempo, o que permitiu a criação de uma comunidade muito grande à volta da mesma existindo assim muita informação disponível.

Os passos tomados na criação da ferramenta estão descritos nas próximas secções.

4.1 Estratégia Utilizada

Para desenhar esta ferramenta foi utilizada a seguinte estratégia: a partir da análise do método escolhido (KAOS) foi desenhado um meta modelo em Ecore que especificasse a sintaxe abstracta dos modelos. A partir deste meta modelo foi criado com recurso à *framework* Eclipse em conjunto com os *plugins* EMF/GMF, um editor de modelos KAOS com uma sintaxe concreta adequada. Nas próximas secções são descritas em maior profundidade os passos e decisões tomados.

4.2 Desenho do modelo Ecore

Para este trabalho foi desenhado um meta modelo baseado em [31]. Foi escolhido este meta modelo porque dentro dos meta modelos encontrados, foi considerado o mais completo, pois suportava todos os conceitos e modelos estudados no âmbito desta dissertação (Modelo de Objectivos, Modelo de Responsabilidades e Modelo de Objectos). Este modelo contém um nó raiz chamado **KAOS** ao qual estão ligados todos os conceitos suportados pelos modelos possíveis de implementar com a ferramenta. Além dos nós que representam os conceitos suportados pelo método, tais como *Objectivo*, *Requisito* ou *Expectativa*, existem também nós para representar ligações entre elementos, como *Refinamento E* ou *Conflito*, e nós para representar os compartimentos (extensão proposta ao método) onde vão ser guardados os conceitos, como por exemplo o *Goal Compartment Node*. Esse meta modelo é apresentado da figura 4.1. Esta figura foi introduzida no relatório para obter uma noção do complexidade do modelo Ecore. Para uma consulta mais pormenorizada, consultar o ficheiro "Ecore.doc". Na secção 4.2.1, é apresentada uma listagem de todos os elementos no modelo Ecore e as relações entre eles.

Para explicar como foram realizados os nós que representam as ligações, vai ser explicada a ligação *Refinamento E* (figura 4.2). Este nó está ligado ao nó raiz através da ligação

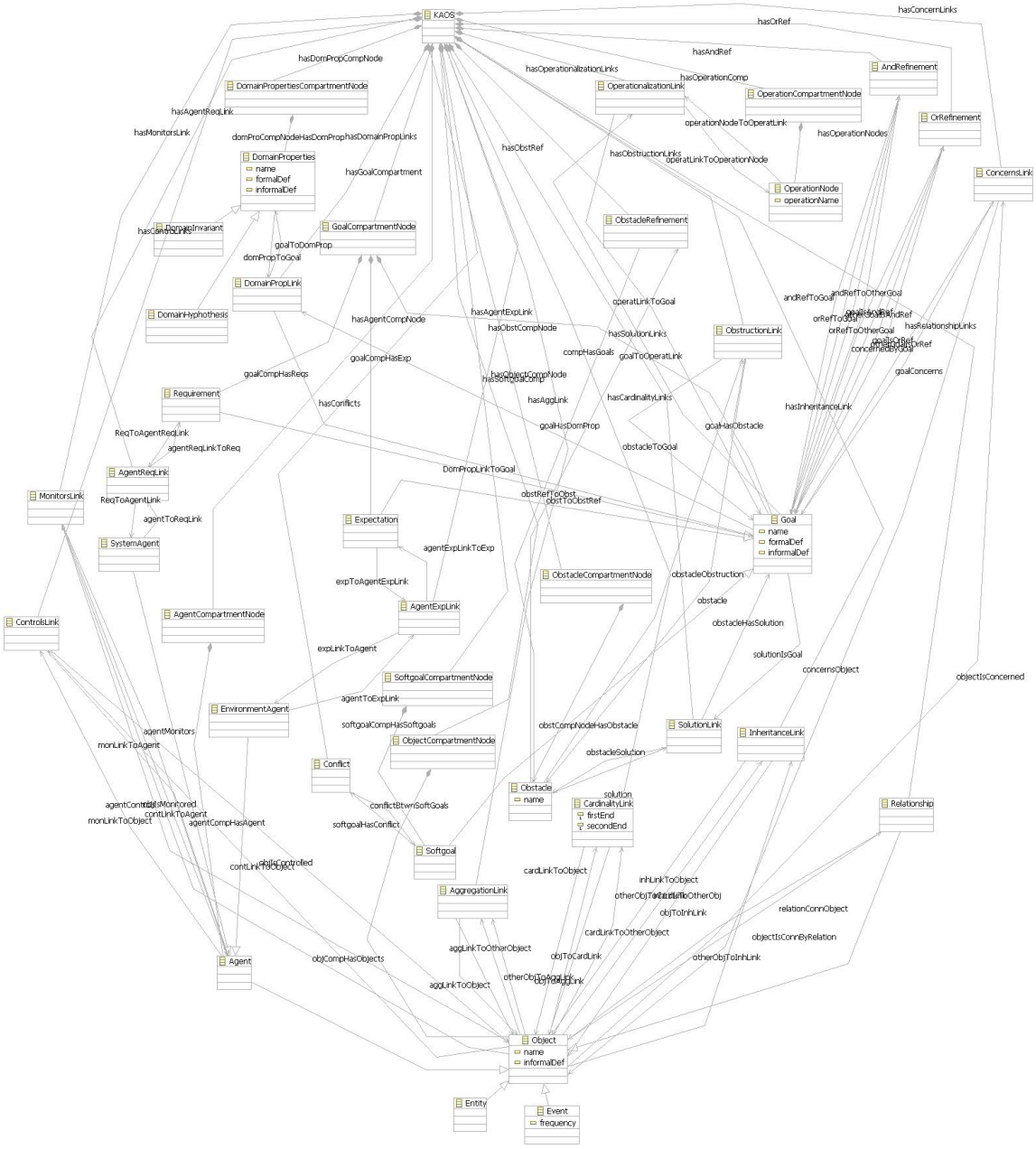

Figura 4.1 Ecore da linguagem implementada.

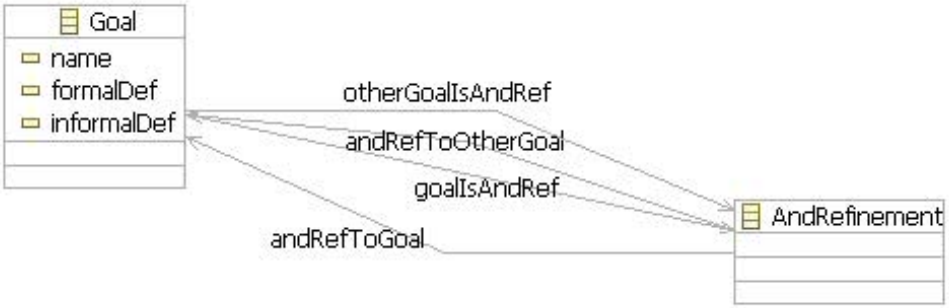

Figura 4.2 Nó Objectivo com Refinamento E

hasAndRef, que significa que os modelos criados com a ferramenta suportam ligações do tipo *Refinamento E*. O nó está também ligado ao nó que representa o conceito *Objectivo* através de quatro ligações: *andRefToGoal*, *andRefToOtherGoal*, *goalIsAndRef* e *otherGoalIsAndRef*. Estas quatro ligações representam a possibilidade de relacionar os Objectivos dos modelos uns com os outros através de ligações Refinamento E. *andRefToGoal* e *goalIsAndRef* são opostos, o que significa que representam a mesma ligação entre dois Objectivos num modelo. A ideia é representar no modelo Ecore uma ligação semelhante às ligações de associação entre classes nos modelos UML, uma vez que a versão EMF usada neste trabalho não suporta ligações bidireccionais. O mesmo acontece às ligações *andRefToOtherGoal* e *otherGoalIsAndRef*.

Esta ferramenta introduz a noção de *Compartimento*. Aqui, um Compartimento é um contentor para elementos de diagramas KAOS. Existem seis tipos de Compartimentos: *GoalCompartmentNode*, para guardar *Objectivos*, *Requisitos* e *Expectativas* (figura 4.3); *SoftgoalCompartmentNode*, para guardar *Requisitos Não-Funcionais* (figura 4.4); *AgentCompartmentNode*, para guardar *Agentes do Sistema* e *Agentes do Ambiente* (figura 4.5); *ObjectCompartmentNode*, para guardar *Entidades* e *Eventos* (figura 4.6); *DomainPropertiesCompartmentNode*, para guardar *Invariantes do Domínio* e *Hipóteses do Domínio* (figura 4.7); *OperationCompartmentNode*, para guardar *Operações* (figura 4.8) e *ObstacleCompartmentNode* para guardar *Obstáculos* (figura 4.9).

Este conceito foi introduzido para ajudar a minimizar os efeitos de escalabilidade nos modelos EROO. Isto é conseguido tornando os compartimentos colapsáveis. Assim, quando um utilizador quer ver apenas uma porção do modelo colapsa os compartimentos correspondentes aos conceitos que pretende esconder.

Durante a definição da sintaxe concreta da linguagem foi decidido que os elementos representativos dos conceitos considerados iriam ter sintaxe semelhante aos elementos utilizados na ferramenta Objectiver, com o objectivo de facilitar o processo de aprendizagem da ferramenta modularKAOS através de associação entre a semelhança da sintaxe dos conceitos, pois a Objectiver é uma ferramenta que já tinha sido previamente utilizada pelo grupo de teste e, sendo uma ferramenta comercial é também muito utilizada no domínio da modelação de diagramas KAOS (ver [11]), logo terá uma grande comunidade de utilizadores conhecedores da sintaxe concreta.

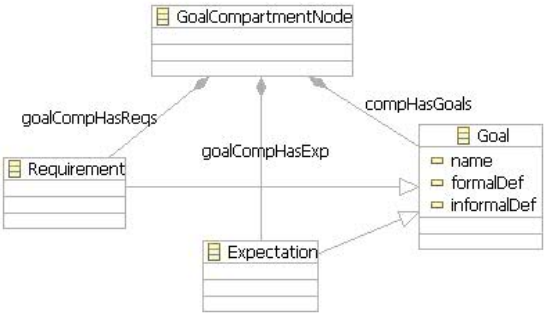

Figura 4.3 Compartimento GoalCompartmentNode

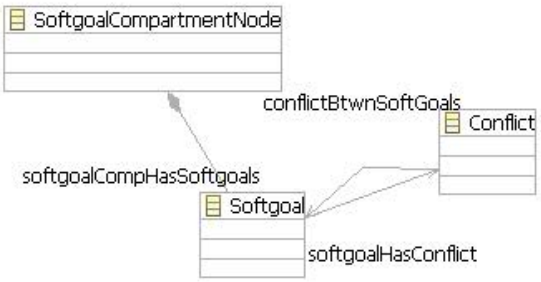

Figura 4.4 Compartimento SoftgoalCompartmentNode

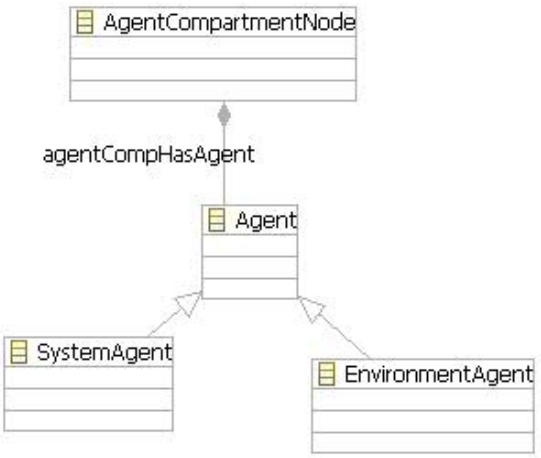

Figura 4.5 Compartimento AgentCompartmentNode

4.2.1 Elementos presentes no modelo Ecore

Nesta subsecção vão ser apresentados todos os conceitos existentes no modelo Ecore, o seu objectivo dentro da linguagem, qual o conceito do KAOS que lhe está associado (caso se aplique)

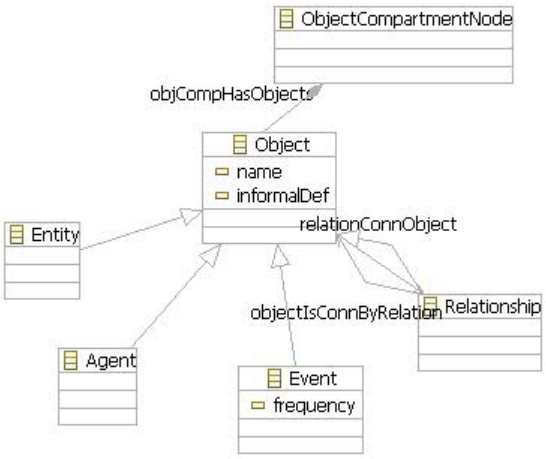

Figura 4.6 Compartimento ObjectCompartmentNode

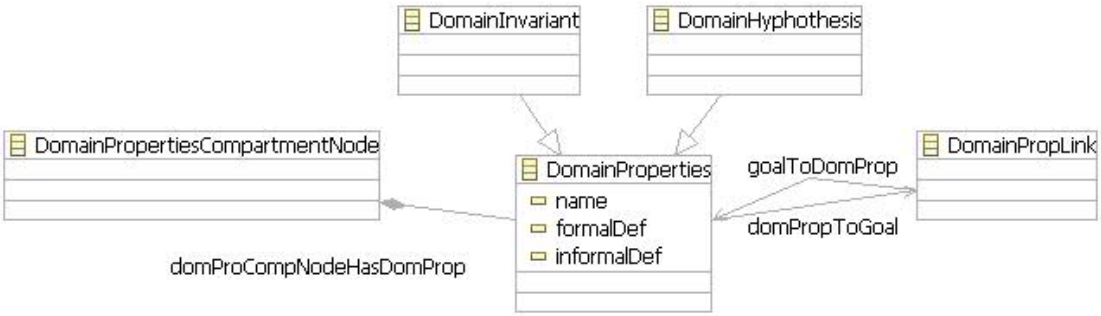

Figura 4.7 Compartimento DomainPropertiesCompartmentNode

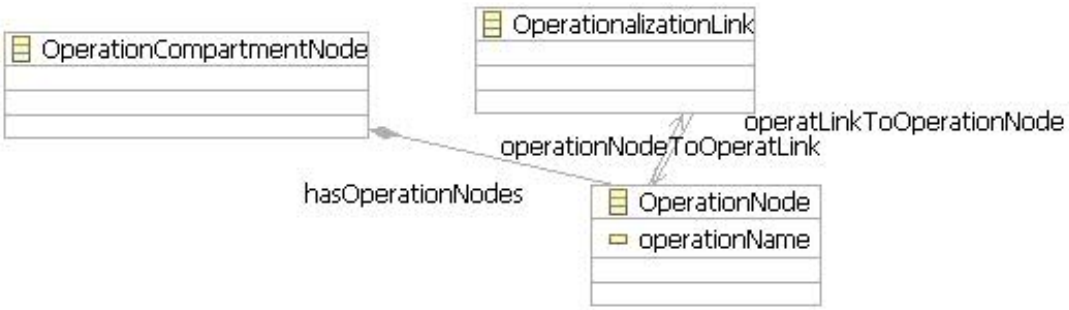

Figura 4.8 Compartimento OperationCompartmentNode

e as respectivas relações entre eles.

- *Goal*: este elemento representa o conceito Objectivo do KAOS. Tem como atributos uma

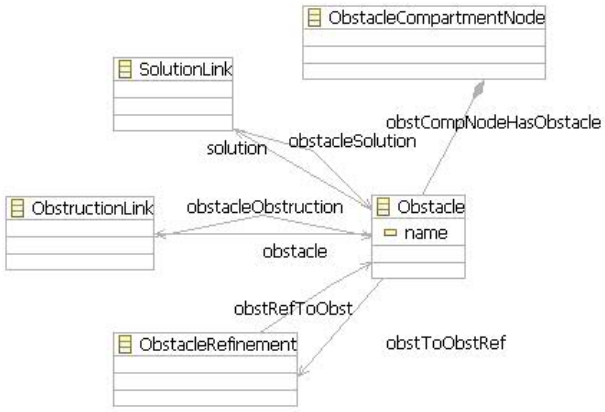

Figura 4.9 Compartimento ObstacleCompartmentNode

String name para representar o nome do Objectivo, uma *String formalDef* que representa a definição formal do Objectivo e uma *String informalDef* que representa a descrição informal. Este conceito tem associados os seguintes pares de ligações (pares esses que seguem a lógica explicada na secção anterior, acerca da ligação *Refinamento E*):

- par *goalHasObstacle-obstacleToGoal*: ligação entre *Goal* e *ObstructionLink*. Este par participa na representação da ligação de obstrução a um Objectivo, isto é, a ligação entre um Objectivo e um Obstáculo;
- par *solutionIsGoal-obstacleHasSolution*: ligação entre *Goal* e *SolutionLink*. Este par vai participar na representação da ligação entre um Obstáculo e o Objectivo que vai servir de solução ao problema apresentado pelo obstáculo;
- par *goalIsAndRef-andRefToGoal*: ligação entre *Goal* e *AndRefinement*. Este par vai participar na representação da possibilidade de criar Refinamentos E entre Objectivos;
- par *otherGoalIsAndRef-andRefToOtherGoal*: mesma ligação que o par anterior. Nesta situação, em que existem dois pares de ligações entre os mesmos dois conceitos, a ideia é indicar que é possível criar Refinamentos E entre dois ou mais Objectivos;
- par *goalIsOrRef-orRefToGoal*: ligação entre *Goal* e *OrRefinement*;
- par *otherGoalIsOrRef-orRefToOtherGoal*: mesma ligação que o par anterior;
- par *goalConcerns-concernedByGoal*: ligação entre *Goal* e *ConcernsLink*;
- par *goalHasDomProp-DomPropLinkToGoal*: ligação entre *Goal* e *DomainPropLink*;
- par *goalToOperatLink-operatLinkToGoal*: ligação entre *Goal* e *Operationalization-Link*.

- *Requirement*: este elemento representa o conceito Requisito do KAOS. Este elemento está relacionado com o elemento *Goal* através de uma relação de herança uma vez que, neste método EROO, os Requisitos são especializações dos Objectivos, se estiverem ligados, neste caso, a um *Agente do Sistema*. Tem associado o seguinte par de ligações:
  - *ReqToAgentReqLink-agentReqLinkToReq*: liga um *Requirement* e um *AgentReqLink*, ou seja, este par vai fazer parte da representação da ligação entre um Agente do Sistema e um Requisito.
- *Expectation*: este elemento representa o conceito Expectativa do KAOS. Tal como o elemento anterior, é uma especialização de *Goal*, por isso está ligado a este elemento através de uma relação de herança. Essa especialização, tal como no método KAOS, é definida pelo Agente ao qual está ligado, neste caso será um *Agente do Ambiente*. Como é uma especialização de *Goal* tem os mesmos atributos, que são, *name*, *formalDef* e *informalDef*. Tem o seguinte par de ligações associado:
  - par *expToAgentExpLink-agentExpLinkToExp*: liga uma *Expectation* e um *AgentExpLink*, ou seja, este par contribui para a criação da ligação entre uma Expectativa e um Agente do Ambiente.
- *Softgoal*: representa o conceito Requisito Não-Funcional do KAOS. Tal como o Requisito e a Expectativa (itens anteriores), é uma especialização do elemento *Goal*. Neste trabalho foi tomada a opção de criar esta especialização dos Objectivos, porque considerou-se que faria mais sentido que a relação de *Conflito*, existente no KAOS, fosse aplicada apenas a Requisitos Não-Funcionais. Então, o elemento *Softgoal* está relacionado com o elemento *Goal*, através de uma relação de herança, tendo por isso os mesmos atributos deste elemento (*name*, *formalDef* e *informalDef*). Este elemento tem o seguinte par de ligações associado:
  - par *softgoalHasConflict-conflictBtwnSoftGoals*: este par liga o elemento *Softgoal* e *Conflict* e participa na criação da ligação de Conflito entre dois Requisitos Não-Funcionais.
- *Conflict*: representa a relação de *Conflito* entre dois Requisitos Não-Funcionais no KAOS. Tal como foi explicado no ponto anterior, na realização desta dissertação considerou-se que este tipo de relação devia acontecer apenas entre Requisitos Não-Funcionais. Tem associado o seguinte par de ligações:
  - par *conflictBtwnSoftGoals-softgoalHasConflict*: este par liga os elementos *Softgoal* e *Conflict* e participa na criação da ligação de *Conflito* entre Requisitos Não-Funcionais, à semelhança do item anterior.
- *AndRefinement*: representa o Refinamento E entre dois ou mais Objectivos no método KAOS. Este elemento representa, não um "elemento físico" como um Objectivo ou um

Requisito, mas sim uma ligação entre elementos, neste caso, elementos do tipo *Goal* e os que com ele têm relações de herança. Para mais informações sobre este tipo de elementos, consultar a explicação associada à figura 4.2, nesta mesma secção. Este elemento apresenta os seguintes pares de ligações:

  - par *andRefToGoal-goalIsAndRef*: este par liga os elementos *Goal* e *AndRefinement* e vai participar na criação do ligações do tipo *Refinamento E* do KAOS;
  - par *andRefToOtherGoal-otherGoalIsAndRef*: este par liga os elementos *Goal* e *AndRefinement* e também vai participar na criação do ligações do tipo *Refinamento E* do KAOS.

- *OrRefinement*: representa o conceito Refinamento Ou entre dois ou mais Objectivos no método KAOS. É um tipo de ligação semelhante ao elemento *AndRefinement* descrito anteriormente, isto é, não representa um conceito "físico" do diagrama mas sim uma ligação entre conceitos do método. Tem associados os seguintes pares de ligações:

  - par *orRefToGoal-goalIsOrRef*: este par de ligações liga os elementos *OrRefinement* e *Goal* e faz parte das ligações que participam na criação da relação *Refinamento Ou* do KAOS, que ligam os Objectivos de um diagrama entre si;
  - par *orRefToOtherGoal-otherGoalIsOrRef*: tal como no ponto anterior, liga os elementos *OrRefinement* e *Goal* e destina-se à criação de ligações do tipo *Refinamento Ou* do KAOS.

- *SystemAgent*: representa um Agente do Sistema do KAOS. Este conceito, tal como no KAOS, é uma especialização de *Agente*. Este elemento, tal como no KAOS, está associado através de uma ligação específica (*AgentReqLink*, que será explicada mais à frente) a um ou mais *Requisitos*. Tem associado o seguinte par de ligações:

  - par *agentToReqLink-ReqToAgentLink*: este par liga os elementos *SystemAgent* e *AgentReqLink* e destina-se a participar na criação da ligação que associa um *Agente do Sistema* e um *Requisito* (ver descrição do elemento *AgentReqLink*).

- *EnvironmentAgent*: representa um Agente do Ambiente do KAOS. Tal como o elemento anterior, é uma especialização de *Agente*, como no método KAOS. Tem associado o seguinte par de ligações:

  - par *agentToExpLink-expLinkToAgent*: liga os elementos *EnvironmentAgent* e *AgentExpLink* e destina-se à criação da ligação entre os elementos *Agente do Ambiente* e *Expectativa* do KAOS (ver descrição do elemento *AgentExpLink*).

- *Agent*: representa o conceito Agente do KAOS. Este elemento, tal como no método KAOS, é uma especialização do conceito *Objecto*, por isso, no modelo Ecore, está relacionado com o elemento *Object* através de uma relação de herança. Tem associados os seguintes pares de ligações:

  - par *agentControls-contLinkToAgent*: liga os elementos *Agent* e *ControlsLink* e destina-se a participar na criação de ligações de *Controlo* entre um *Agente* e um *Objectivo*;
  - par *agentMonitors-contLinkToAgent*: liga os elementos *Agent* e *MonitorsLink* e serve na criação da ligação de *Monitorização* entre um *Agente* e *Objectivo*.

- *Obstacle*: representa o conceito Obstáculo do método KAOS. Tem um atributo *name*, para indicar o nome do Obstáculo, do tipo *String*. Tem associados os seguintes pares de ligações:

  - par *solution-obstacleSolution*: liga os elementos *Obstacle* e *SolutionLink*. Este par de ligações participa na criação da ligação de *Solução* entre um *Obstáculo* e um *Objectivo*, isto é, o Objectivo ligado ao Obstáculo com esta ligação é solução do mesmo;
  - par *obstacleObstruction-obstacle*: liga os elementos *Obstacle* e *ObstructionLink*.

- *Object*: representa um Objecto do método KAOS. Tem como atributos uma *String name*, que representa o nome do Objecto, e uma *String informalDef*, que representa a descrição informal do Objecto. Este elemento tem associados os seguintes pares de ligações:

  - par *objectIsConnByRelation-relationConnObject*: este par participa na criação da ligação de *Relação* do KAOS entre dois Objectos;
  - par *objectIsConcerned-concernsObject*: este par liga os elementos *Object* e *ConcernsLink*, destinando-se a participar na criação da ligação de *Concerns* que relaciona um Objecto com um Objectivo para o qual esse Objecto é importante;
  - par *objIsControlled-contLinkToObject*: este par liga os elementos *Object* e *ControlLinks*, para participar na criação da ligação de Controlo entre um Objecto e um Agente no método KAOS;
  - par *objIsMonitored-monLinkToObject*: este par liga os elementos *Object* e *MonitorsLink* e servirá para criar a ligação de Monitorização entre um Agente e um Objecto no método KAOS;
  - par *objToCardLink-cardLinkToObject*: este par liga os elementos *Object* e *CardinalityLink* para participar na criação da ligação de Cardinalidade entre dois Objectos;
  - par *otherObjToCardLink-cardLinkToOtherObject*: este par liga os elementos *Object* e *CardinalityLink* e, juntamente com o par mencionado no ponto anterior, vai participar na criação da relação de Cardinalidade entre dois Objectos;
  - par *objToInhLink-inhLinkToObject*: este par liga os elementos *Object* e *InheritanceLink*, tendo como objectivo participar na criação de uma relação de Herança entre dois ou mais Objectos;

  - par *otherObjToInhLink-inhLinkToOtherObj*: este par está relacionado com o par anterior, liga os elementos *Object* e *InheritanceLink* e vai participar na criação da ligação de Herança entre dois ou mais Objectos;
  - par *objToAggLink-aggLinkToObject*: este par liga os elementos *Object* e *AggregationLink* e participa na criação da ligação de Agregação entre dois ou mais Objectos.
  - par *otherObjToAggLink-aggLinkToOtherObject*: este par liga os elementos *Object* e *AggregationLink*. Este par está relacionado com o par anterior na criação da ligação de Agregação entre dois ou mais Objectos do método KAOS.

- *Event*: representa um Evento do método KAOS. Este elemento tem um atributo *frequency* do tipo *Int* para registar a frequência da ocorrência do Evento. Como Event é uma especialização de *Object* (através de uma relação de herança), tem além do atributo *frequency*, os atributos *name* e *informalDef*.

- *Entity*: representa um Entidade do método KAOS. Tem os mesmos atributos de *Object*, pois é uma especialização deste elemento.

- *Relationship*: representa um Relacionamento entre *Objectos* do método KAOS. Este conceito do KAOS representa uma ligação entre dois Objectos e é uma especialização do elemento *Object*. Tem o seguinte par de ligações associado:

  - par *relationConnObject-objectIsConnByRelation*: este par liga os elementos *Object* e *Relationship* e destina-se a participar na criação da ligação de Relação entre dois Objectos do método KAOS.

- *DomainProperties*: representa Propriedades do Domínio do método KAOS. Este elemento tem os atributos *name* do tipo *String*, para guardar o nome, *formalDef* do tipo *String*, para guardar a descrição formal, e *informalDef* do tipo *String*, para guardar a descrição informal do elemento. Tem associado o seguinte par de ligações:

  - par *domPropToGoal-goalToDomProp*: este par liga os elementos *DomainProperties* e *DomainPropLink* e participa na criação da ligação que relaciona um Objectivo com uma Propriedade do Domínio.

- *DomainInvariant*: representa uma Invariante do Domínio do KAOS. Trata-se de uma especialização da *Propriedade do Domínio*, estando por isso ligado ao elemento correspondente a esse conceito através de uma relação de herança e tem os mesmos atributos desse elemento.

- *DomainHypothesis*: representa uma Hipótese do Domínio do KAOS. Tal como a *Invariante do Domínio*, é uma especialização da *Propriedade do Domínio*, por isso estes elementos estão relacionados através de uma relação de herança no modelo Ecore e tem os mesmos atributos da Propriedade do Domínio.

- *KAOS*: este elemento do modelo Ecore não representa um conceito do método KAOS, mas sim um "diagrama", isto é, todos os elementos que lhe estão ligados são elementos possíveis de criar no editor da ferramenta modularKAOS. Este elemento tem as seguintes ligações associadas:

  - *hasAndRef*: liga os elementos *KAOS* e *AndRefinement*. A existência desta ligação indica que os diagramas criados nesta ferramenta permitem a criação de ligações Refinamento E;
  - *hasOrRef*: liga os elementos *KAOS* e *OrRefinement*. Esta ligação está relacionada com a possibilidade de criar ligações de Refinamento Ou com a ferramenta;
  - *hasConflicts*: liga os elementos *KAOS* e *Conflict*. Esta ligação está relacionada com a criação de ligações de Conflito com a ferramenta;
  - *hasAgentReqLink*: liga os elementos *KAOS* e *AgentReqLink*. Esta ligação indica que é possível criar ligações entre *Requisitos* e *Agentes do Sistema*;
  - *hasAgentExpLink*: liga os elementos *KAOS* e *AgentExpLink*. Esta ligação indica que é possível criar ligações entre *Expectativas* e *Agentes do Ambiente*;
  - *hasObstructionLinks*: liga os elementos *KAOS* e *ObstructionLink*. Esta ligação está relacionada com a possibilidade de criar ligações de Obstrução entre um *Objectivo* e um *Obstáculo*;
  - *hasSolutionLinks*: liga os elementos *KAOS* e *SolutionLink*. Esta ligação está relacionada com a possibilidade de criar ligações de Solução entre um *Objectivo* e um *Obstáculo*;
  - *hasDomainPropLinks*: liga os elementos *KAOS* e *DomainPropLink*. Esta ligação está relacionada com a ligação que relaciona uma Propriedade do Domínio e um Objectivo;
  - *hasConcernsLinks*: liga os elementos *KAOS* e *ConcernsLink*. Esta ligação está ewalcionada com a ligação que relaciona um Objectivo e um Objecto;
  - *hasRelationshipLinks*: liga os elementos *KAOS* e *Relationship*. Esta ligação está relacionada com o conceito de Relação do método KAOS;
  - *hasControLinks*: liga os elementos *KAOS* e *ControlsLink*. Está relacionado com a ligação de Controlo entre um Agente e um Objecto;
  - *hasMonitorsLink*: liga os elementos *KAOS* e *MonitorsLink*. Está relacionado com a ligação de Monitorização entre um Agente e um Objecto;
  - *hasSoftgoalComp*: liga os elementos *KAOS* e *SoftgoalCompartmentNode*. Esta ligação está relacionada com a possibilidade de criar compartimentos para guardar Requisitos Não-Funcionais do KAOS;

  - *hasGoalCompartment*: liga os elementos *KAOS* e *GoalCompartmentNode*. Esta ligação está relacionada com a possibilidade de criar compartimentos para guardar Objectivos, Requisitos e Expectativas do KAOS;
  - *hasObstCompNode*: liga os elementos *KAOS* e *ObstacleCompartmentNode*. Esta ligação está relacionada com a possibilidade de criar compartimentos para guardar Obstáculos do KAOS;
  - *hasDomPropCompNode*: liga os elementos *KAOS* e *DomainPropertiesCompartmentNode*. Esta ligação está relacionada com a possibilidade de criar compartimentos para guardar Invariantes do Domínio e Hipóteses do Domínio;
  - *hasAgentCompNode*: liga os elementos *KAOS* e *AgentCompartmentNode*. Esta ligação está relacionada com a possibilidade de criar compartimentos para guardar Agentes do Sistema e Agentes do Ambiente;
  - *hasObjectCompNode*: liga os elementos *KAOS* e *ObjectCompartmentNode*. Esta ligação está relacionada com a possibilidade de criar compartimentos para guardar Eventos e Entidades do KAOS;
  - *hasOperationComp*: liga os elementos *KAOS* e *OperationCompartmentNode*. Esta ligação está relacionada com a possibilidade de criar compartimentos para guardar Operações do método KAOS;
  - *hasOperationalizationLinks*: liga os elementos *KAOS* e *OperationalizationLink*. Esta ligação está relacionada com o conceito de ligação de Operacionalização do método KAOS;
  - *hasCardinalityLinks*: liga os elementos *KAOS* e *CardinalityLink*. Esta ligação está relacionada com a ligação de Cardinalidade do método KAOS;
  - *hasInheritanceLink*: liga os elementos *KAOS* e *InheritanceLink*. Esta ligação está relacionada com a ligação de Herança do método KAOS;
  - *hasAggLink*: liga os elementos *KAOS* e *AggregationLink*. Esta ligação está relacionada com a ligação de Agregação do método KAOS;
  - *hasObstRef*: liga os elementos *KAOS* e *ObstacleRefinement*. Esta ligação está relacionada com a ligação de refinamento de um Obstáculo.

- *AgentReqLink*: representa a ligação entre um *Requisito* e um *Agente do Sistema* no método KAOS. Tem associados os seguintes pares de ligações:

  - par *agentReqLinkToReq-ReqToAgentReqLink*: este par liga os elementos *AgentReqLink* e *Requirement* e participa na criação da ligação que une um *Agente do Sistema* a um *Requisito*;
  - par *ReqToAgentLink-agentToReqLink*: este par liga os elementos *AgentReqLink* e *SystemAgent*. Esta ligação participa na criação da ligação entre um *Agente do Sistema* e um *Requisito*.

- *AgentExpLink*: representa uma ligação entre uma *Expectativa* e um *Agente do Ambiente* no método KAOS. Tem associados os seguintes pares de ligações:
  - par *agentExpLinkToExp-expLinkToAgent*: liga os elementos *AgentExpLink* e *Expectation*. Está relacionado com a criação de uma ligação entre uma Expectativa e um Agente do Ambiente;
  - par *expLinkToAgent-agentToExpLink*: liga os elementos *AgentExpLink* e *EnvironmentAgent*. Participa na criação da ligação que afecta um Agente do Ambiente a uma Expectativa.
- *ObstructionLink*: representa a relação entre um *Objectivo* e um *Obstáculo* que lhe esteja associado. Tem associados os seguintes pares:
  - par *obstacle-obstacleObstruction*: liga os elementos *ObstructionLink* e *Obstacle*. Vai fazer parte da criação da ligação de Obstrução entre um *Obstáculo* e um *Objectivo*;
  - par *obstacleToGoal-goalHasObstacle*: liga os elementos *ObstructionLink* e *Goal*. Faz parte da criação da ligação de Obstrução entre um Obstáculo e um Objectivo.
- *SolutionLink*: esta ligação vai ligar o *Objectivo* que servirá de *Solução* ao *Obstáculo* com quem está relacionado. Tem associados os seguintes pares de ligações:
  - par *obstacleHasSolution-solutionIsGoal*: liga os elementos *SolutionLink* e *Goal*. Faz parte da criação da ligação de Solução entre um Obstáculo e um Objectivo;
  - par *obstacleSolution-solution*: liga os elementos *SolutionLink* e *Obstacle*. Faz parte da criação da ligação de Solução entre um Objectivo e um Obstáculo.
- *ConcernsLink*: representa a ligação que relaciona um *Objecto* com um *Objectivo*. Tem associados os seguintes pares de ligações:
  - par *concernsObject-objectIsConcerned*: liga os elementos *ConcernsLink* e *Object*. Este par participa na criação da ligação que relaciona um Objecto com um Objectivo que precisa desse Objecto para a sua realização;
  - par *concernedByGoal-goalConcerns*: liga os elementos *ConcernsLink* e *Goal*. Participa na criação da ligação que relaciona um Objectivo e um Objectivo. Está relacionado com o par anterior.
- *DomainPropLink*: representa a ligação que relaciona uma *Propriedade do Domínio* e um *Objectivo*. Tem associados os seguintes pares:
  - par *goalToDomProp-domPropToGoal*: liga os elementos *DomainPropLink* e *DomainProperties*;

  - par *DomPropLinkToGoal-goalHasDomProp*: liga os elementos *DomainPropLink* e *Goal*.

- *MonitorsLink*: representa a ligação de Monitorização que liga um *Agente* e um *Objecto*. Tem associados os seguintes pares de ligações:

  - par *monLinkToObject-objIsMonitored*: liga os elementos *MonitorsLink* e *Object*;
  - par *monLinkToAgent-agentMonitors*: liga os elementos *MonitorsLink* e *Agent*.

- *ControlsLink*: representa a ligação de Controlo que liga um *Agente* e um *Objecto*. Tem associados os seguintes pares de ligações:

  - par *contLinkToObject-objIsControlled*: liga os elementos *ControlsLink* e *Object*;
  - par *contLinkToAgent-agentControls*: liga os elementos *ControlsLink* e *Agent*.

- *GoalCompartmentNode*: faz parte do grupo de extensões feitas à linguagem. Este elemento representa um *Compartimento* e destina-se a guardar *Objectivos*, *Requisitos* e *Expectativas*. Tem um atributo *name*, do tipo *String*, para guardar o nome do Compartimento. Tem associadas as seguintes ligações:

  - *compHasGoals*: liga o compartimento ao elemento *Goal*. Esta ligação representa a possibilidade de colocar Objectivos dentro do Compartimento;
  - *goalCompHasReqs*: liga o compartimento ao elemento *Requirement*. Esta ligação representa a possibilidade de colocar Requisitos dentro do Compartimento;
  - *goalCompHasExp*: liga o compartimento ao elemento *Expectation*. Esta ligação representa a possibilidade de colocar Expectativas dentro do Compartimento.

- *SoftgoalCompartmentNode*: parte das extensões propostas para o meta modelo do KAOS, é um *Compartimento* para guardar *Requisitos Não-Funcionais*. Tem um atributo *name*, do tipo *String*, para guardar o nome. Tem associada a seguinte ligação:

  - *softgoalCompHasSoftgoals*: liga o compartimento e o elemento *Softgoal*. Esta ligação representa a possibilidade de criar Requisitos Não-Funcionais dentro deste compartimento.

- *ObstacleCompartmentNode*: este *Compartimento* destina-se a guardar *Obstáculos* do método KAOS. Tem um atributo *name*, do tipo *String*, para guardar o nome. Tem associada a seguinte ligação:

  - *obstCompNodeHasObstacle*: liga o compartimento e o elemento *Obstacle*. Esta ligação representa a possibilidade de colocar Obstáculos no compartimento.

- *DomainPropertiesCompartmentNode*: neste Compartimento vai ser possível guardar *Invariantes do Domínio* e *Hipóteses do Domínio*. Tem um atributo *name*, do tipo *String*, para guardar o nome do compartimento. Tem associada a seguinte ligação:
  - *domProCompNodeHasDomProp*: liga o compartimento ao elemento *DomainProperties*. Representa a possibilidade de colocar Hipóteses do Domínio e Invariantes do Domínio dentro do Compartimento.
- *AgentCompartmentNode*: neste Compartimento é possível colocar *Agentes do Sistema* ou *Agentes do Ambiente*. Tem um atributo *String name*, para guardar o nome do compartimento. Tem associada a seguinte ligação:
  - *agentCompHasAgent*: liga o compartimento ao elemento *Agent*. A existência desta ligação significa que é possível colocar Agentes do Sistema ou Agentes do Ambiente dentro do compartimento.
- *ObjectCompartmentNode*: este Compartimento permite guardar *Entidades* ou *Eventos*. Tem um atributo *name*, do tipo *String*, para guardar o nome. Tem associada a seguinte ligação:
  - *objCompHasObjects*: liga o compartimento ao elemento *Object*. A sua existência significa que é possível colocar Eventos ou Entidades dentro do compartimento.
- *OperationCompartmentNode*: neste Compartimento é possível guardar *Operações* do método KAOS. Tem um atributo *name*, do tipo *String*, para guardar o nome. Tem associada a seguinte ligação:
  - *hasOperationNodes*: liga o compartimento ao elemento *OperationNode*. Representa a possibilidade de colocar Operações dentro do compartimento.
- *OperationNode*: este elemento representa o conceito Operação do método KAOS. Tem um atributo *operationName* do tipo *String*, que representa o nome da Operação representada no modelo. Tem associado o seguinte par de ligações:
  - *operationNodeToOperatLink-operatLinkToOperationNode*: este par liga os elementos *OperationNode* e *OperationalizationLink* e participa na criação da ligação de Operacionalização, que une uma Operação a um Objectivo.
- *OperationalizationLink*: esta elemento representa a ligação de Operacionalização que relaciona um *Objectivo* com uma *Operação*. Tem associados os seguintes pares de ligações:
  - par *operatLinkToOperationNode-operationNodeToOperatLink*: liga os elementos *OperationalizationLink* e *OperationNode*. Participa na criação da ligação de Operacionalização, entre uma Operação e um Objectivo;

  - par *operatLinkToGoal-goalToOperatLink*: liga os elementos *Operationalization-Link* e *Goal* e participa na criação da ligação de Operacionalização, que relaciona um Objectivo com uma Operação.

- *CardinalityLink*: este elemento representa uma ligação de Cardinalidade entre dois *Objectos*. Tem associados os seguintes pares de ligações:

  - par *cardLinkToObject-objToCardLink*: une os elementos *CardinalityLink* e *Object*. Faz parte da criação da ligação de Cardinalidade entre dois objectos;
  - par *cardLinkToOtherObject-otherObjToCardLink*: une os elementos *Cardinality-Link* e *Object*.

- *InheritanceLink*: este elemento representa uma ligação de Herança entre duas ou mais *Entidades*. Tem associados os seguintes pares de ligações:

  - par *inhLinkToObject-objToInhLink*: liga os elementos *InheritanceLink* e *Object*;
  - par *inhLinkToOtherObj-otherObjToInhLink*: liga os elementos *InheritanceLink* e *Object*.

- *AggregationLink*: este elemento representa uma ligação de Agregação entre dois ou mais *Objectos*. Tem associados os seguintes pares de ligações:

  - par *aggLinkToObject-objToAggLink*: liga os elementos *AggregationLink* e *Object*;
  - par *aggLinkToOtherObject-otherObjToAggLink*: liga os elementos *AggregationLink* e *Object*.

- *ObstacleRefinement*: esta ligação serve para refinar *Obstáculos* do KAOS em vários sub-Obstáculos. Tem associada a seguinte ligação:

  - *obstRefToObst*: liga os elementos *ObstacleRefinement* e *Obstacle*.

4.3 A Ferramenta

A figura 4.10 mostra o editor da ferramenta modularKAOS. Este apresenta um espaço branco onde vão ser criados os modelos e um menu (assinalado a vermelho), onde estão presentes os elementos necessários para a criação de diagramas: Compartimentos (*Compartments*), Ligações (*Links*) e Nós (*Nodes*).

A figura 4.11 mostra mais em pormenor o menu de selecção.

Com a modularKAOS é possível criar *Modelos de Objectivos* (figuras 4.12 e 4.13), *Modelos de Responsabilidade* (figura 4.14) e *Modelos de Objectos* (figura 4.15). Na figura 4.12 é mostrado um objectivo principal que é depois decomposto em outros sub-objectivos, que por sua vez são decompostos em requisitos ou expectativas. Neste modelo é também possível ver

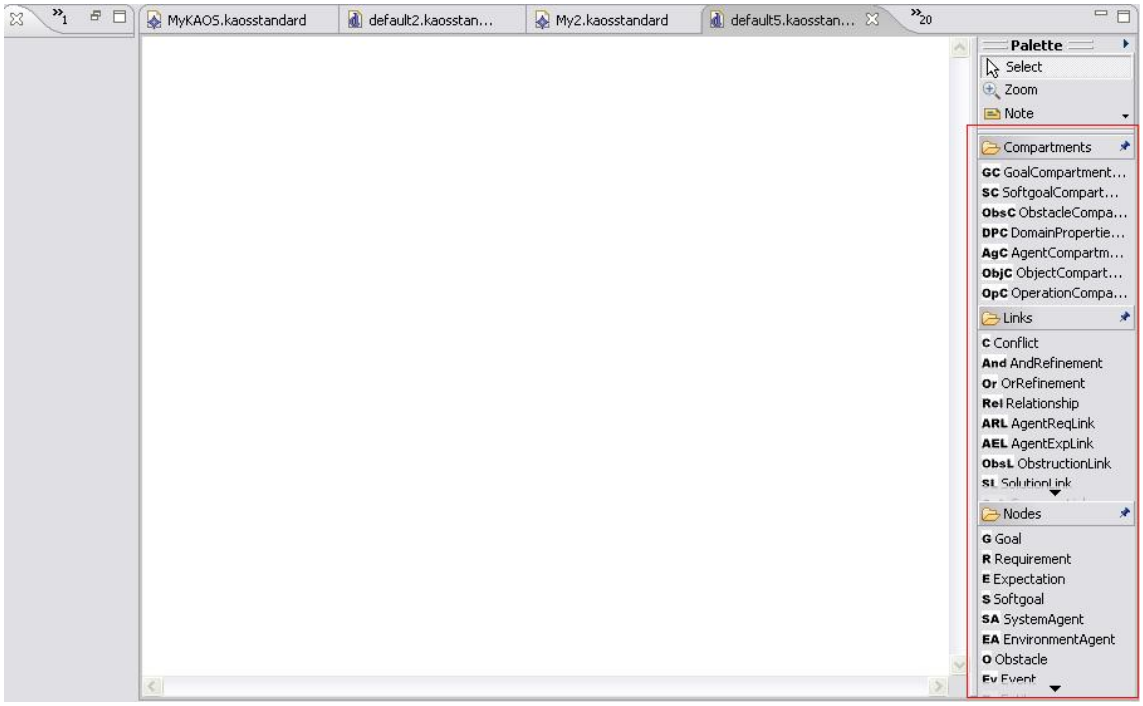

Figura 4.10 Editor da ferramenta.

Figura 4.11 Menu de selecção da ferramenta.

um obstáculo a um objectivo, com a respectiva solução, assim como alguns requisitos não funcionais. A figura 4.13 representa o mesmo diagrama da figura 4.12, com a diferença de ter alguns

compartimentos colapsados. Na figura 4.14 vê-se um agente (mais concretamente um Agente do Sistema) que é responsável por vários requisitos. Na figura 4.15 vêem-se vários objectos que estão ligados entre si através de ligações de agregação e herança.

Para se criar um modelo qualquer com a ferramenta, deve-se colocar primeiro um compar-

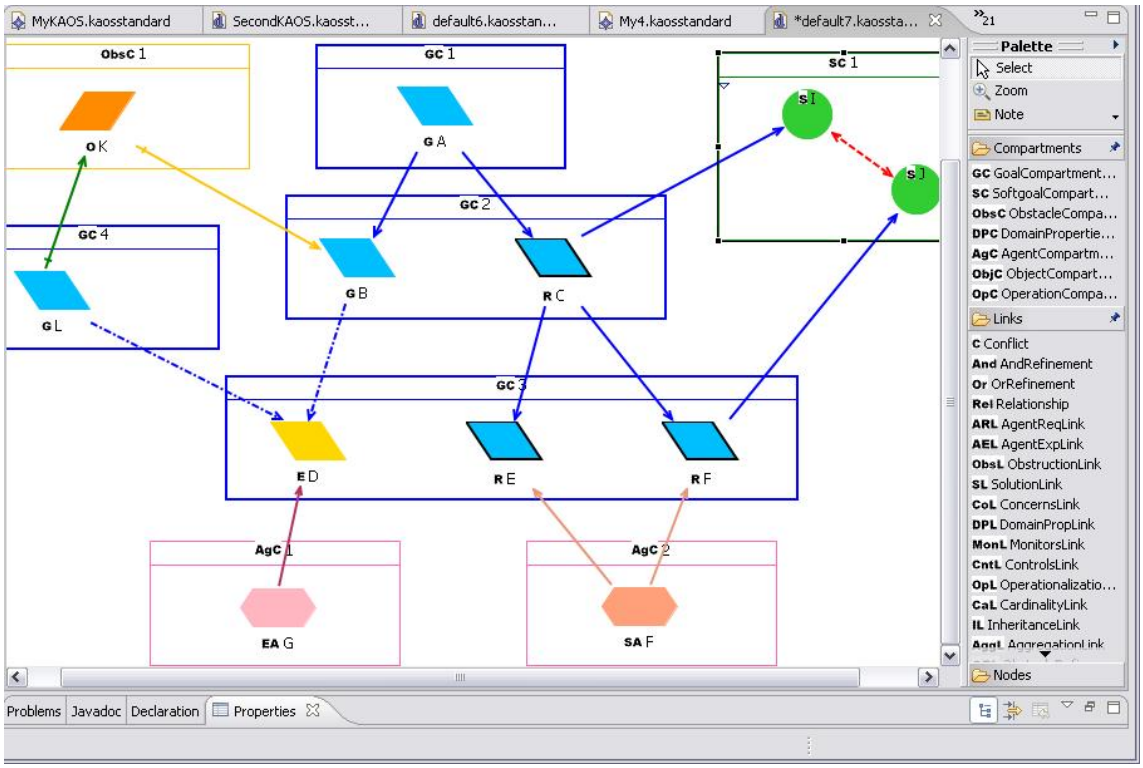

Figura 4.12 Modelo de Objectivos realizado com a ferramenta.

timento no espaço de edição (por exemplo, um *GoalCompartmentNode*). Depois colocam-se dentro do compartimento quais os conceitos desejados, de acordo com o tipo de compartimento escolhido (por exemplo, dentro de um *GoalCompartmentNode* só é possível colocar *Objectivos*, *Requisitos* ou *Expectativas*). De seguida, criam-se as ligações desejadas entre elementos. É possível ligar elementos entre diferentes compartimentos, desde que sejam ligações sintacticamente correctas de acordo com o meta modelo definido. Para mais informações sobre como utilizar a ferramenta, ver o apêndice B.

A tabela 4.1 apresenta os elementos suportados pela ferramenta.

4.4 Restrições OCL

Além das restrições impostas pelo meta modelo criado, foram também criadas algumas restrições OCL nomeadamente ao nível de algumas ligações entre elementos. A restrição introduzida

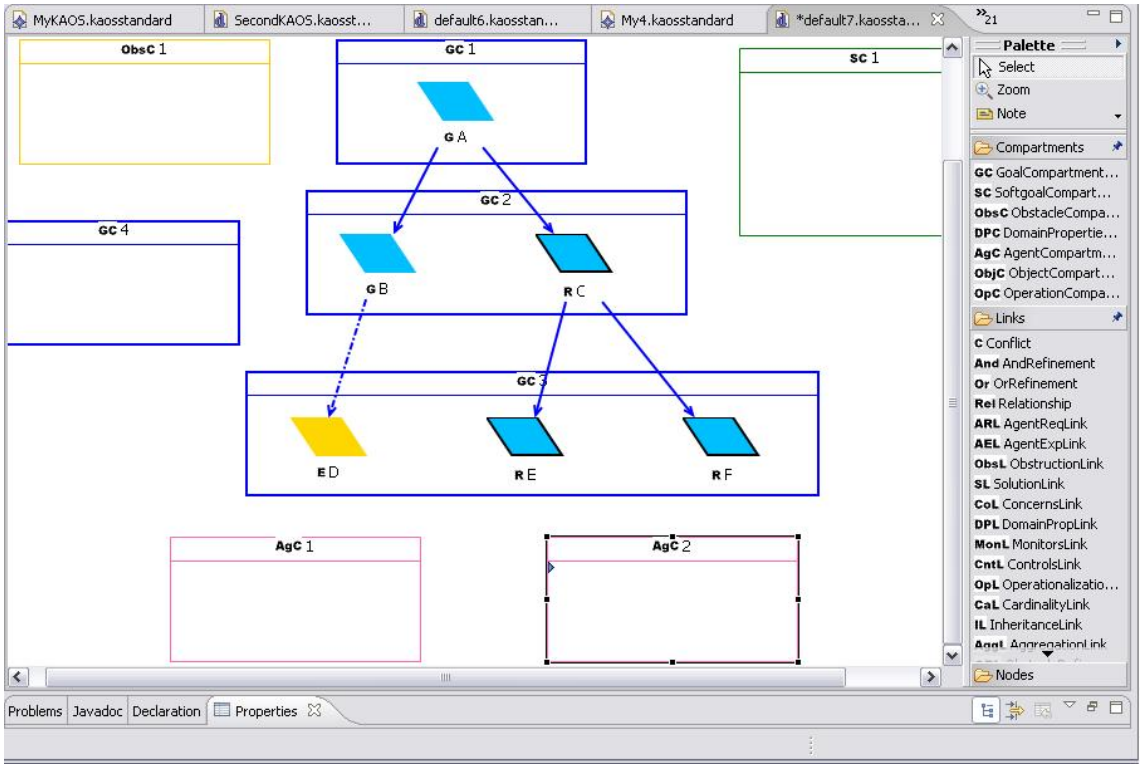

Figura 4.13 Modelo de Objectivos com alguns compartimentos colapsados.

foi feita para evitar ligações de um elemento para ele próprio quando esta não fazia sentido, como por exemplo, um Objectivo refinar-se a ele próprio. Para isso, a expressão OCL usada foi *self* <> *oppositeEnd*, isto é, o ponto inicial e o ponto final de uma ligação não podem coincidir no mesmo elemento. Esta restrição foi feita nas ligações possíveis de criar entre elementos do mesmo tipo, isto é, Agregação, Refinamento E, Relação de Cardinalidade, Conflito, Herança, Refinamento Ou, Relação entre Objectos e Refinamento de Obstáculos.

4.5 Sumário do Capítulo

Neste capítulo foram descritos os passos de criação da ferramenta, nomeadamente qual a estratégia utilizada para abordar o problema, o desenho do modelo Ecore e quais os resultados obtidos, mais concretamente o editor que foi criado e quais os conceitos que implementa.

No próximo capítulo vai ser descrito como foi feita a validação dos resultados.

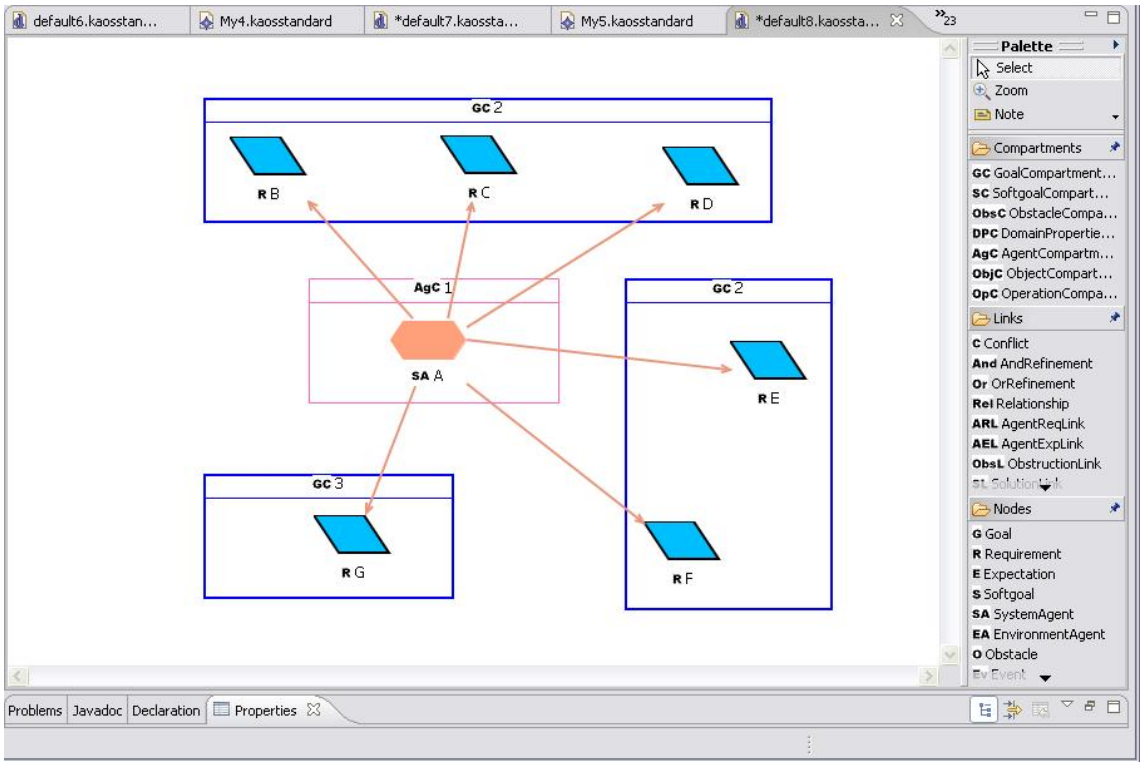

Figura 4.14 Modelo de Responsabilidades.

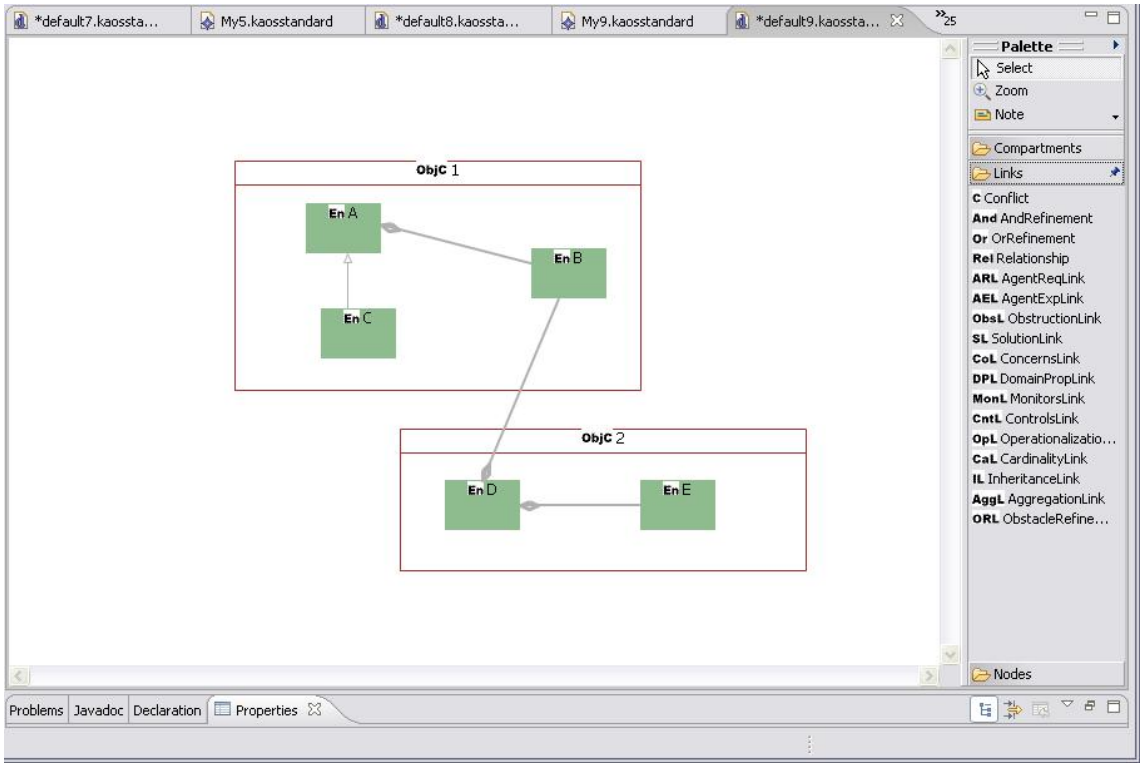

Figura 4.15 Modelo de Objectos.

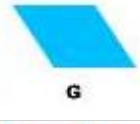	Objectivo
	Requisito
	Expectativa
	Requisito Não-Funcional
	Agente do Ambiente
	Agente do Sistema
	Obstáculo
	Operação
	Evento
	Entidade
	Hipótese do Domínio
	Invariante do Domínio

Tabela 4.1 Conceitos suportados pela ferramenta.

5. Validação

A validação feita teve como principais objectivos avaliar a Expressividade e Usabilidade da ferramenta modularKAOS. Para validar a Expressividade foi utilizado o caso de estudo do sistema BART (referido na secção 2.2) para verificar se a sintaxe abstracta definida no meta modelo se mantinha nos modelos criados com a ferramenta. Neste caso de estudo foram focados aspectos relacionados com a velocidade e aceleração dos comboios no sistema [22, 37]. Foi também usado o caso de estudo do SAP (ver secção 5.3). Para avaliar a Usabilidade foi feito um inquérito, com questões sobre a sintaxe da linguagem, facilidade de utilização da ferramenta e níveis de satisfação com a ferramenta, a um grupo de nove alunos com conhecimento do método KAOS e que já tinham experimentado outra ferramenta de modelação (Objectiver), para além da desenvolvida no âmbito desta dissertação, através da frequência de uma cadeira da área de Engenharia de Requisitos. A parte da validação relativa à Usabilidade é descrita em mais pormenor na secção 5.1.

5.1 Questionário

O inquérito realizado foi baseado em [33]. Existiam questões de interpretação e de resposta aberta e um problema para resolver com a ferramenta. O primeiro grupo de questões (secção A.2) foi sobre interpretação da linguagem: eram apresentados aos utilizadores alguns modelos (Modelo de Objectivos, Modelo de Responsabilidades e Modelo de Objectos), semelhantes aos apresentados nas figuras 4.12, 4.14 e 4.15 e os utilizadores tinham de identificá-los, assim como os conceitos presentes neles, feita através da ligação entre o conceito questionado e o respectivo nome. O segundo grupo de questões (secção A.3) era para desenhar, com a ferramenta modularKAOS, um modelo previamente criado com o Objectiver, na sequência da cadeira referida anteriormente. Esse problema era sobre o abastecimento de um veículo automóvel numa bomba de gasolina que estava inserida no sistema Via Verde, onde o utilizador para poder abastecer o veículo tinha de ter um identificador válido e realizar várias operações. O terceiro grupo (secção A.4) tinha a ver com o que os utilizadores acharam da ferramenta. A classificação da ferramenta era feita com recurso a uma escala de cinco valores, que variavam entre "Muito Adequado/Mais Fácil/Mais Simples/Melhor" e "Muito Inadequado/Mais Difícil/Mais Complexo/Pior" (os valores apresentados variavam conforme a questão feita). No final existiam também algumas questões de resposta aberta onde os utilizadores davam a sua opinião sobre os pontos fracos e fortes da ferramenta testada. Para analisar o questionário com mais detalhe, este pode ser consultado no apêndice A.

5.2 Resultados da Validação

Os utilizadores mostraram grande aceitação da ferramenta. Em geral ficaram satisfeitos com ela e acharam-na fácil de usar, comparativamente com ferramentas usadas anteriormente. A noção de Compartimento foi muito bem aceite, devido ao uso potencial para esconder a escalabilidade dos modelos através da colapsação das caixas, além de ajudar a estruturar os modelos desenhados. Vários utilizadores disseram que a ferramenta era intuitiva e fácil de utilizar ("(...) *pode ser muito intuitiva e fácil de usar* (...)", "(...) *Fácil de utilizar* (...) *Os menus são intuitivos e fáceis de utilizar* (...)") e um mencionou que tem "(...) *uma curva de aprendizagem bastante suave (...)* ". Alguns utilizadores mencionaram também que a ferramenta tinha boa interface visual.

Em relação à primeira questão, o modelo melhor identificado foi o Modelo de Objectos, tendo sido identificado correctamente por todos os utilizadores (tabela 5.1).

De um grupo de onze conceitos, uma média de nove foi identificada correctamente (tabela 5.2).

Os conceitos melhor identificados foram *Requisito* e *Agregação* (foram identificados por

Modelo	**Respostas Correctas**
Modelo de Objectivos	8
Modelo de Objectos	**9**
Modelo de Responsabilidades	8

Tabela 5.1 Modelos identificados pelos utilizadores.

Número do Questionário	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
Número de Conceitos Correctos	8	8	9	6	9	11	8	10	10
Média	9								

Tabela 5.2 Número de conceitos correctos por questionário

todos os utilizadores). O conceito pior identificado foi *Refinamento Ou* (apenas cinco utilizadores identificaram-no correctamente) (tabela 5.3). Em relação à terceira questão, as respostas dadas estão sumariadas da figura 5.1 à 5.8. Na escala usada para representar as respostas dos utilizadores, "1" representa o valor melhor (por exemplo, "Mais Fácil") e "5" representa o valor pior (por exemplo, "Mais Difícil") da escala.

Devido ao número pequeno de pessoas que realizou o teste, os resultados apresentados são apenas indicativos da possível aceitação da ferramenta para criar modelos orientados a objectivos. Este pequeno grupo de utilizadores mostrou uma resposta positiva à ferramenta e no futuro espera-se que possa ser testada por um grupo maior e mais heterogéneo de pessoas para que os resultados possam ser melhor validados.

A figura 5.1 compara a adequação da sintaxe usada pelas duas ferramentas comparadas

Conceito	Respostas Correctas
1.A - *Agente*	6
1.F - *Requisito*	**9**
2.B - *Entidade*	7
Ligação 2.A - 2.B - *Agregação*	**9**
3.J - *Requisito Não-Funcional*	7
3.D - *Expectativa*	8
3.L - *Objectivo*	7
3.K - *Obstáculo*	8
Ligação 3.A - 3.B - *Refinamento E*	7
Ligação 3.L - 3.D - *Refinamento Ou*	**5**
Ligação 3.I - 3.J - *Conflito*	6

Tabela 5.3 Conceitos identificados pelos utilizadores.

(modularKAOS e Objectiver). Em geral, a sintaxe foi considerada adequada para as duas ferramentas com uma ligeira vantagem para a ferramenta modularKAOS. Em relação à facilidade de interpretação de modelos criados com as respectivas ferramentas (figura 5.2), os utilizadores acharam mais fácil interpretar modelos criados com a modularKAOS. A figura 5.3 compara o

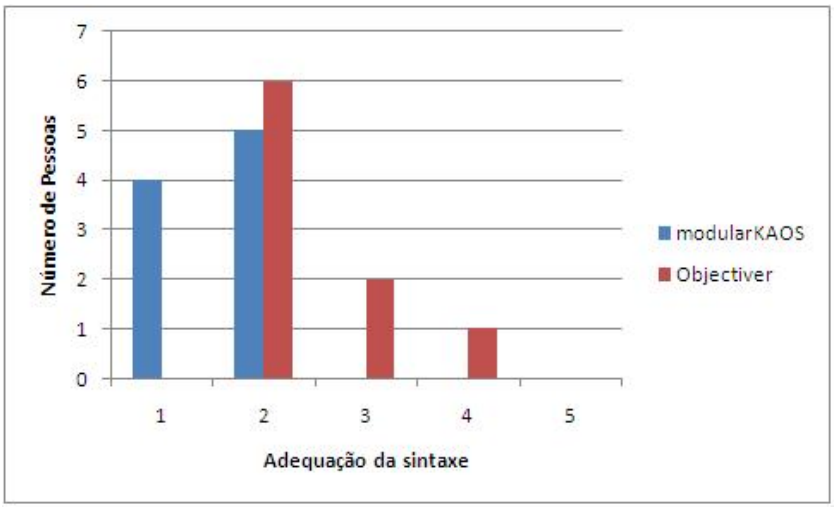

Figura 5.1 Sintaxe da linguagem.

que os utilizadores achavam ao nível da facilidade da interpretação de modelos comparativamente com outras ferramentas conhecidas. Os utilizadores consideraram que, em comparação com outras ferramentas previamente utilizadas, os modelos criados com a modularKAOS são mais fáceis de interpretar. Na figura 5.4 compara a facilidade em criar modelos com as ferramentas usadas pelos utilizadores. Em geral, os utilizadores consideraram que era mais fácil

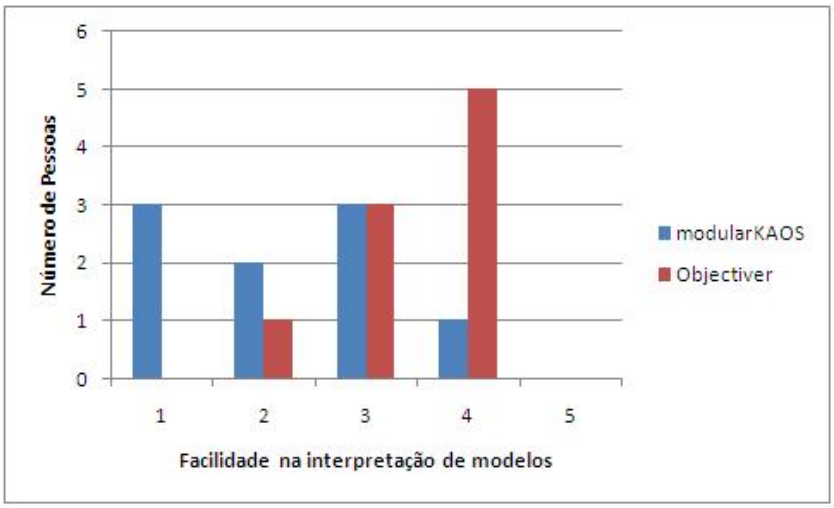

Figura 5.2 Facilidade de interpretação dos modelos.

criar modelos com a ferramenta modularKAOS. Na figura 5.5 são mostrados os resultados

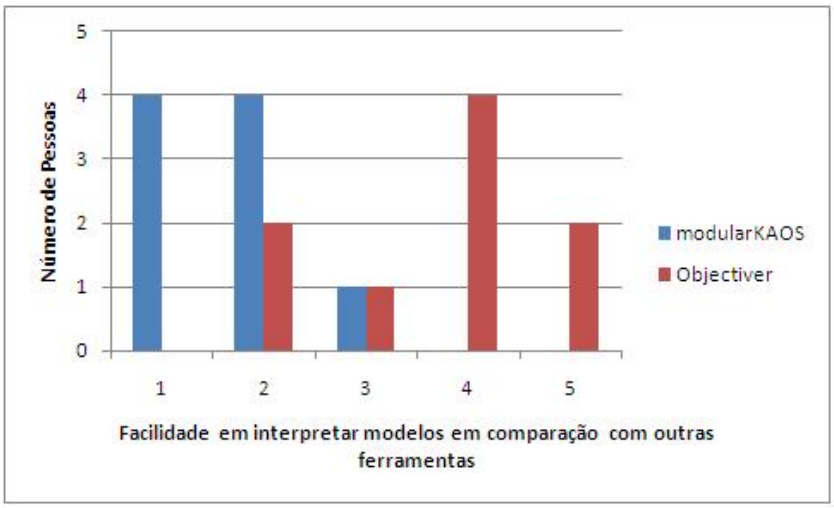

Figura 5.3 Facilidade de interpretação comparada com outras ferramentas.

referentes à pergunta sobre o nível de facilidade de criação de modelos com as ferramentas em análise, comparativamente com outras ferramentas previamente conhecidas. A maioria dos utilizadores achou mais fácil criar modelos com o modularKAOS em comparação com outras ferramentas previamente conhecidas. Em relação ao que os utilizadores achavam sobre a maneira como as ferramentas lidam com o problema da escalabilidade em comparação com outras ferramentas conhecidas, a grande maioria considerou que a modularKAOS lidava melhor com o problema de escalabilidade de modelos, tal como é mostrado na figura 5.6. A figura 5.7 apre-

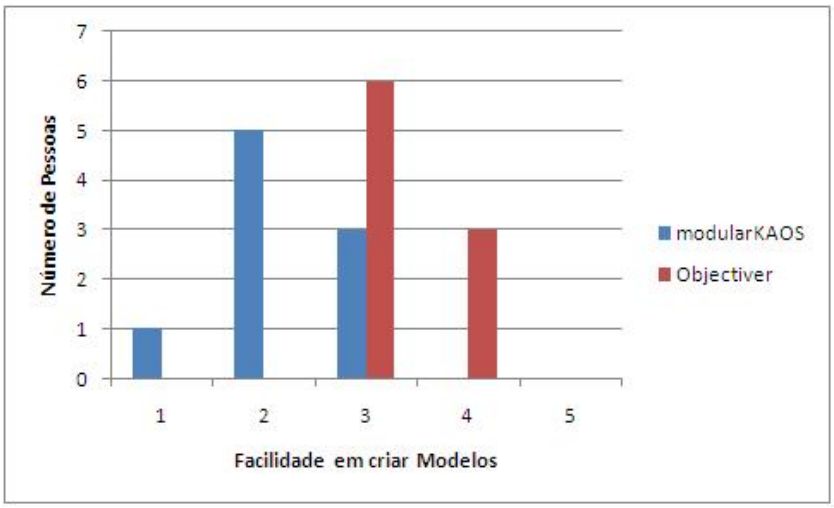

Figura 5.4 Facilidade em criar modelos.

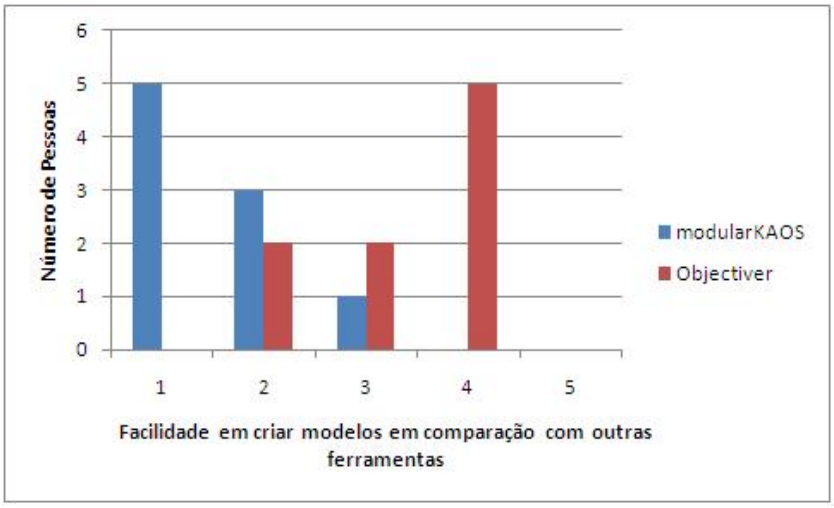

Figura 5.5 Facilidade em criar modelos comparado com outras ferramentas.

senta as respostas dos utilizadores sobre a eficiência das ferramentas na criação de modelos, comparativamente com outras ferramentas previamente conhecidas. Os utilizadores consideraram a modularKAOS mais eficiente na criação de modelos em relação a outras ferramentas previamente conhecidas. Finalmente, na figura 5.8 são apresentados os resultados em relação à questão sobre se as ferramentas testadas trouxeram ou não alguma inovação ao método KAOS. Em relação à ferramenta modularKAOS as respostas foram consensuais pois a maioria dos utilizadores considerou que esta ferramenta trouxe muita inovação ao método. Em relação à ferramenta Objectiver as respostas foram mais distribuídas, não sendo possível obter um padrão de respostas.

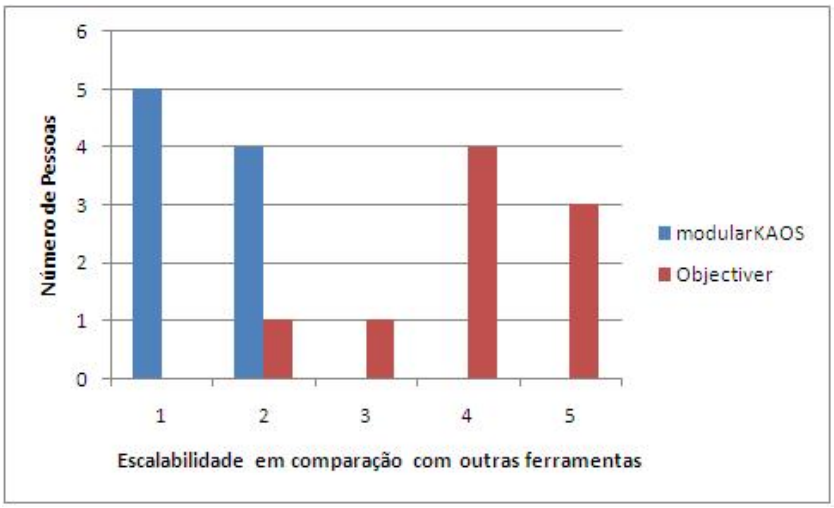

Figura 5.6 Modo como a ferramenta lida com a escalabilidade de modelos.

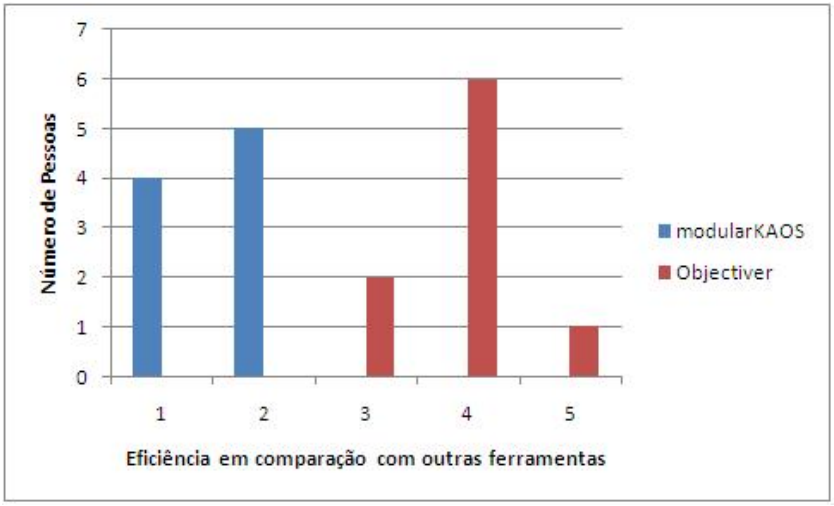

Figura 5.7 Eficiência da Ferramenta.

5.3 Caso de Estudo do SAP

Para validação adicional da ferramenta, foi escolhido um caso de estudo do SAP sobre linhas de produto de software no contexto de aplicações de negócio. Foi retirado do site do projecto AMPLE [15], sendo o exemplo de um caso industrial realizado na vida real. Este caso de estudo tem como foco a gestão de relação entre clientes (em inglês, CRM - **C**ustomer **R**elationship **M**anagement). São consideradas quatro documentos de especificações de requisitos: "*Market Requirements Document*", que descreve três áreas particulares e três "*Requirements Specifications*", que descrevem três áreas no cenário de vendas, que são:

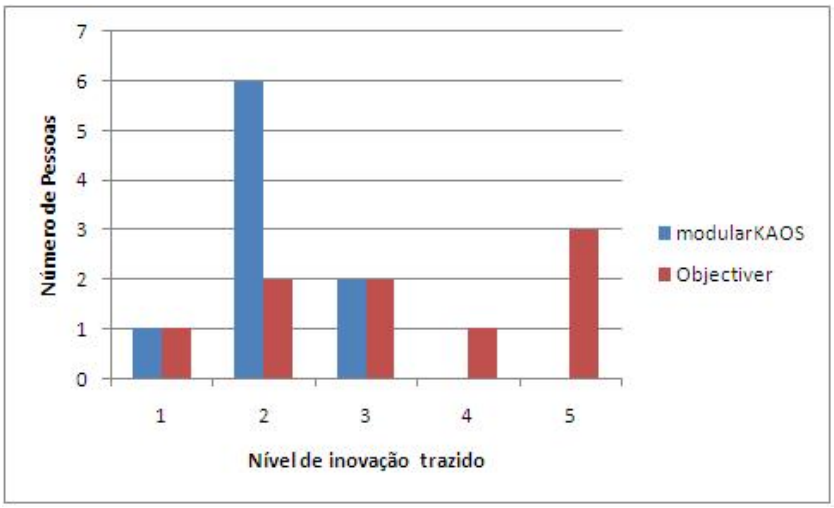

Figura 5.8 Inovações ao método trazidas pela ferramenta.

- **Gestão das Encomendas dos Clientes** (*Customer Order Management*): envolve os seguintes requisitos: (i) Processamento de Vendas, (ii) Cotações, (iii) Atribuição de Preços, (iv) Processo de Aprovação, (v) Verificação de Disponibilidade, (vi) Verificação do Crédito, (vii) Devoluções;

- **Pagamentos** (*Payment*): gere os pagamentos recebidos e em dívida. É activado automaticamente após a criação de uma ordem de venda. Existem três meios de pagamento: (i) Pagamento por Cartão - é debitada automaticamente a quantia da conta do utilizador, (ii) Pagamento em Dinheiro - a factura pode ser anexada à encomenda, (iii) Pagamento por Factura (requisito opcional) - é enviada uma factura ao cliente e é fornecida a opção de liquidá-la mais tarde;

- **Gestão de Produtos** (*Product Management*): gere os produtos em stock. Compreende os seguintes requisitos: (i) Gestão de Stocks Simples - trata do caso em que os produtos são mantidos num único armazém, tendo como principal preocupação garantir que existe uma relação directa entre o inventário dos produtos e a capacidade suportada. Deve também ser possível gerir os meta dados relativos à localização dos produtos; (ii) Gestão de Stocks Múltiplos - trata do caso em que os produtos são mantidos em vários armazéns, tendo como principal preocupação calcular o inventário total. Deve também calcular quais os armazéns mais perto da morada de entrega da encomenda; (iii) Gestão de Produtos Complexos - os produtos são compostos por outros produtos e o sistema deve calcular o inventário total desse produto composto, tendo em conta as peças constituintes mesmo que estejam em armazéns dispersos; (iv) Integração da Gestão de Produção - fornece funções de automatização e monitorização tal que, quando o produto não exista em stock, deve ser encomendado tendo em conta a disponibilidade financeira actual e a

quantidade necessária.

5.3.1 Modelação com a *modularKAOS*

Foi criado um modelo que abrange os três cenários na secção anterior. Nesta sub-secção são apresentadas várias perspectivas do modelo: perspectiva geral e uma perspectiva para cada um dos três cenários.

5.3.1.1 Perspectiva geral do modelo

Na figura 5.9 é possível ver o modelo com os três cenários todos visíveis. Pode ser consultada uma versão maior da figura no ficheiro anexo "SAP.doc".

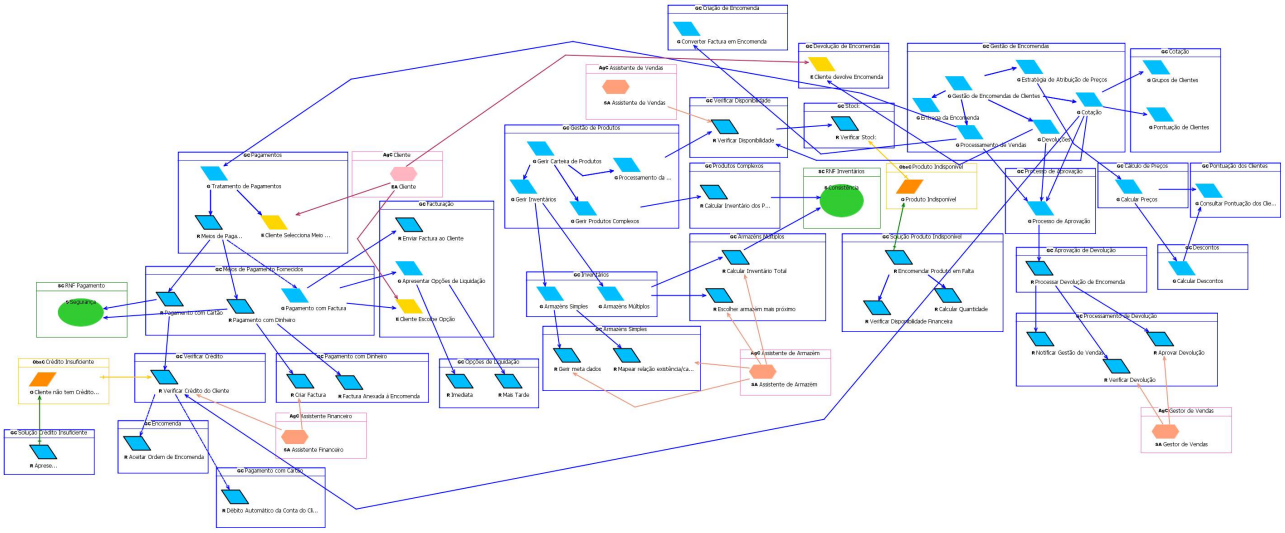

Figura 5.9 Perspectiva Geral do Caso de Estudo do SAP.

5.3.1.2 Cenário 1 - Gestão de Encomendas dos Clientes

Na figura 5.10 a opção tomada foi visualizar o cenário "Gestão de Encomendas dos Clientes". Para isso, colapsou-se os compartimentos correspondentes aos outros dois cenários ("Gestão de Produtos" e "Pagamentos").

5.3.1.3 Cenário 2 - Pagamentos

Na figura 5.11 a opção tomada foi visualizar o cenário "Pagamentos". Para isso, colapsou-se os compartimentos correspondentes aos outros dois cenários ("Gestão de Produtos" e "Gestão de Encomendas dos Clientes").

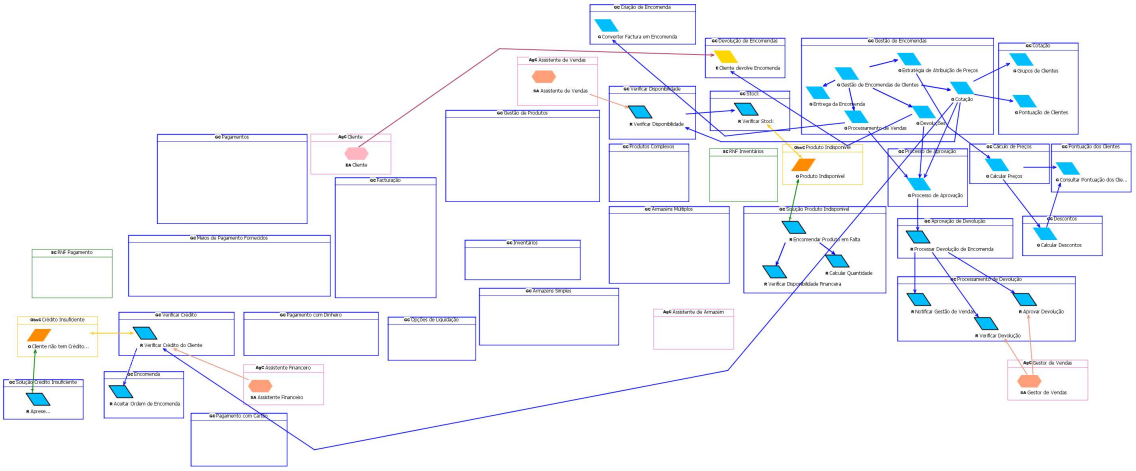

Figura 5.10 Visualização do Cenário “Gestão de Encomendas de Clientes”.

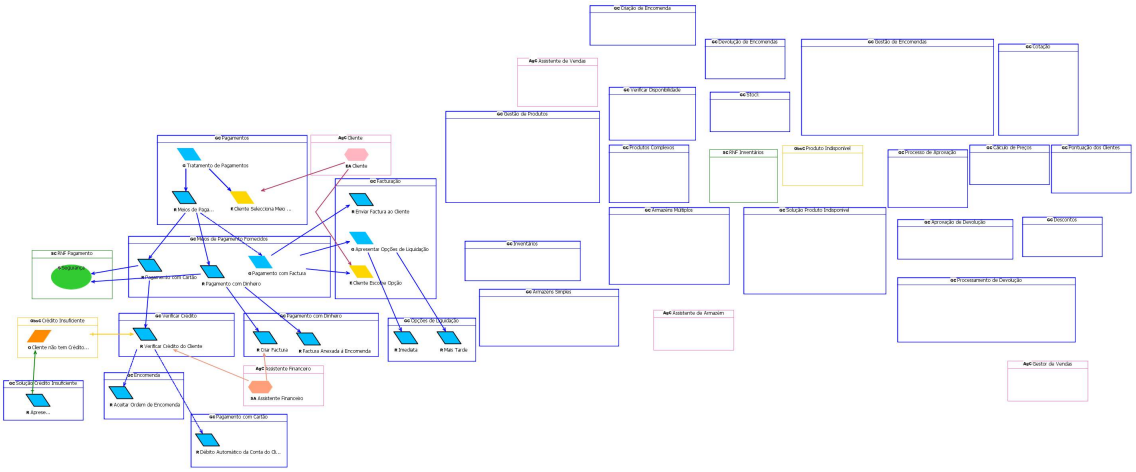

Figura 5.11 Visualização do Cenário “Pagamentos”.

5.3.1.4 Cenário 3 - Gestão de Produtos

Na figura 5.12 a opção tomada foi visualizar o cenário “Gestão de Produtos”. Para isso, colapsou-se os compartimentos correspondentes aos outros dois cenários (“Gestão de Produtos” e “Gestão de Encomendas dos Clientes”).

5.4 Sumário do Capítulo

Neste capítulo é descrito como foi feita a validação da ferramenta, ao nível de Expressividade e Usabilidade. Para abordar o primeiro ponto usámos um caso de estudo de complexidade média

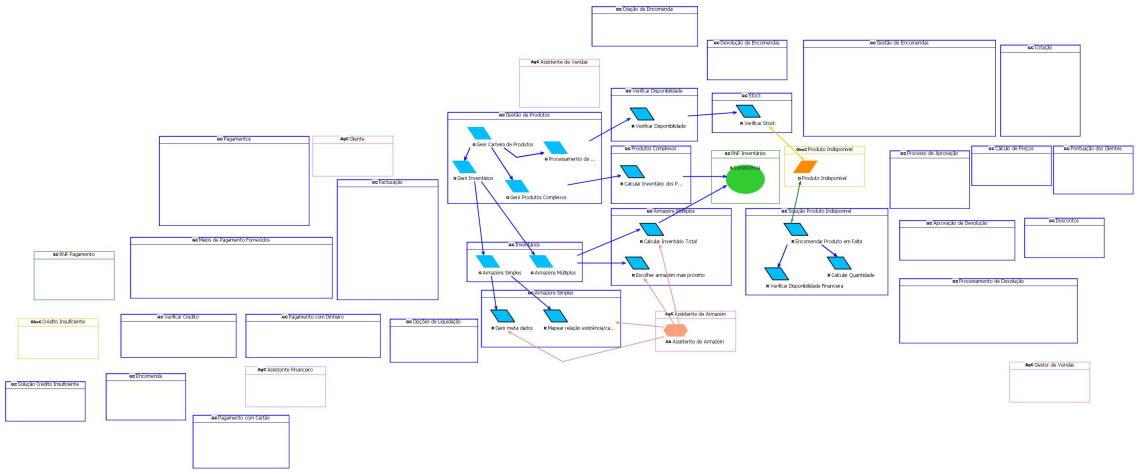

Figura 5.12 Visualização do Cenário “Gestão de Produtos”.

e avaliámos se as novas construções na linguagem inseriram limitações na expressividade. Para abordar o segundo ponto, realizámos testes de usabilidade com um grupo de utilizadores.

No próximo capítulo são apresentadas as conclusões finais a retirar do trabalho.

6. Conclusão

Nesta dissertação foi estudado um método EROO, mais precisamente o método KAOS, e foi desenhado um novo meta modelo da linguagem baseado num previamente existente, introduzindo a noção de Compartimento como inovação ao método. Este Compartimento é colapsável e, na sua forma colapsada, esconde o que está no seu interior o que pode ser útil quando se pretende analisar apenas porções do modelo em estudo, além de melhorar a complexidade visual dos mesmos. A partir do meta modelo estendido foi criada uma Linguagem Específica do Domínio assim como um editor associado de modo a ser possível criar modelos KAOS com as extensões propostas. Após a fase de implementação, foram realizados alguns testes à ferramenta com um pequeno grupo de utilizadores conhecedores do método KAOS e que haviam usado previamente uma outra ferramenta de modelação, além da criada no âmbito deste trabalho. Chegou-se então a uma ferramenta concreta que, de acordo com as indicações, é usável e expressiva com uma sintaxe abstracta que verifica os modelos criados.

A noção de Compartimento foi muito bem aceite por todos os utilizadores, que consideraram um conceito com um potencial de utilidade muito grande uma vez que, devido à capacidade de colapsação referida, permite lidar melhor com os problemas de escalabilidade em diagramas muito grandes em comparação com outras ferramentas utilizadas anteriormente. As conclusões que se podem tirar deste trabalho são apenas indicativas, pois o grupo de teste à ferramenta foi muito pequeno (apenas nove utilizadores). As respostas à ferramenta foram muito positivas ao nível dos testes de Usabilidade (ver a secção 5.2).

Espera-se no futuro poder realizar testes a um grupo mais alargado e heterogéneo de utilizadores para poder comprovar os resultados obtidos pelo primeiro grupo de utilizadores.

6.1 Trabalho Futuro

A partir do trabalho realizado no âmbito desta dissertação podem ser estudados outros métodos EROO, como por exemplo i*, de modo a encontrar aspectos em comum entre os dois métodos de modo a no futuro construir eventualmente um novo método híbrido e mais eficiente. Estes estudos sobre os aspectos comuns entre os métodos podem também servir de base para estudos iniciais na área de transformações entre modelos (transformar modelos KAOS para i* e vice-versa).

A . Questionário sobre a LED KAOS

A.1 Introdução

O objectivo deste questionário é avaliar a usabilidade de uma ferramenta para criação de modelos KAOS.

O questionário está dividido em três partes. Na primeira parte são apresentados alguns modelos realizados com a ferramenta onde se pretende avaliar a semântica e capacidade de abstração dos modelos criados. Na segunda parte apresentam-se alguns problemas que deverão ser modelados utilizando a ferramenta. A terceira parte são algumas questões para avaliar o nível de satisfação dos utilizadores.

Desde já obrigada pela colaboração.

A.2 Interpretação de modelos

- Considere as seguintes figuras:

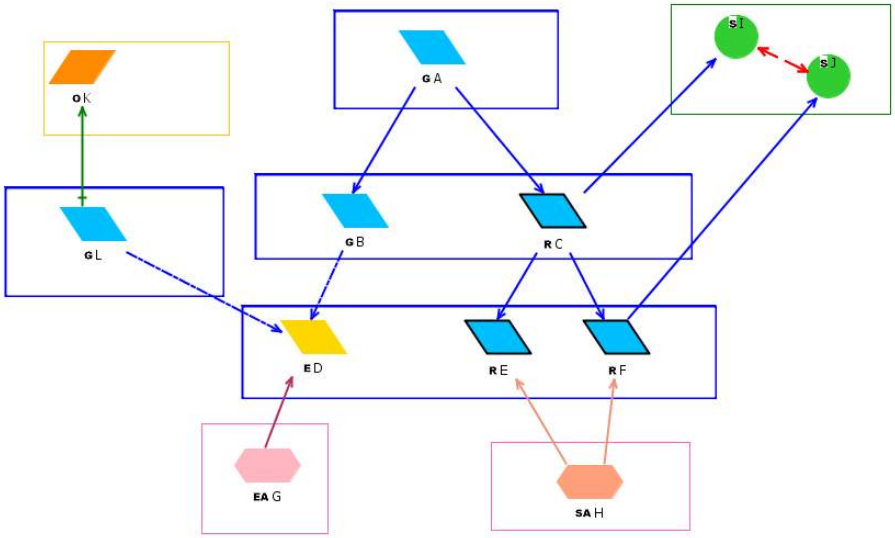

Figura A.1 Figura 1.

A que modelo correspondem cada uma das figuras?
Modelo de Objectivos ____
Modelo de Responsabilidades ____
Modelo de Objectos ____

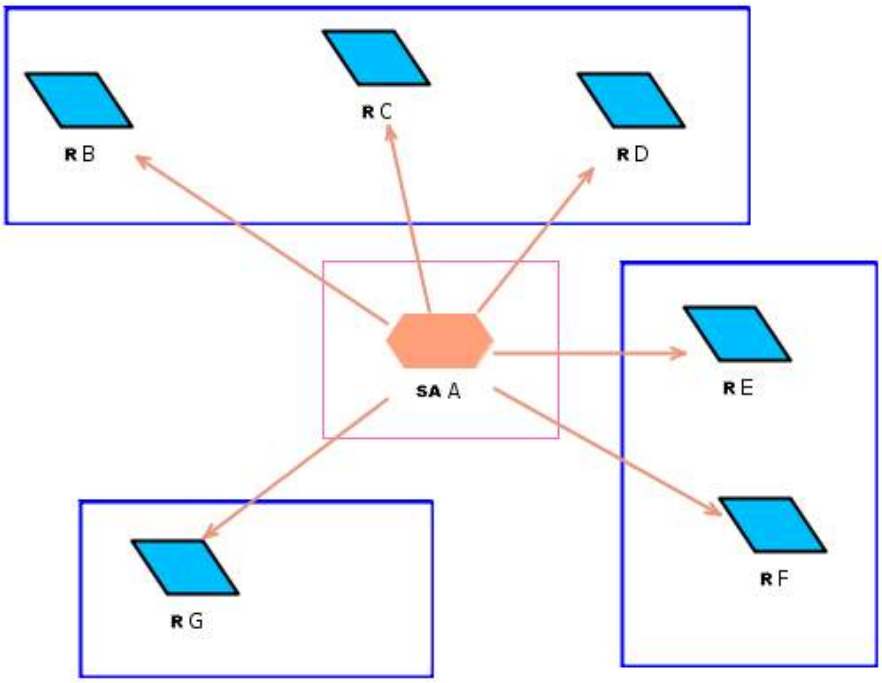

Figura A.2 Figura 2.

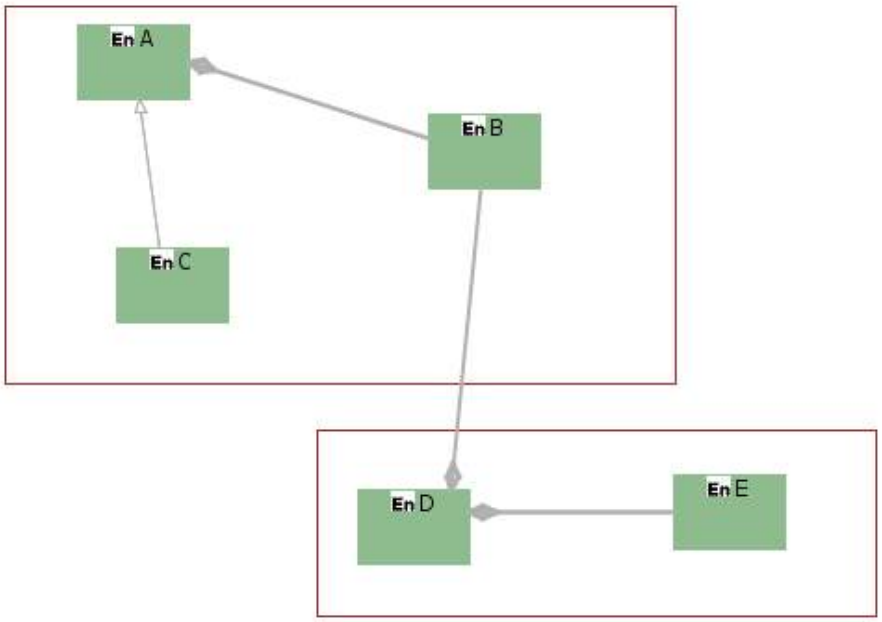

Figura A.3 Figura 3.

- Quais os conceitos representados pelos elementos? (Nota: elemento 1.A, por exemplo, significa o elemento A da figura 1)

1.A	Objectivo
1.F	Requisito
2.B	Expectativa
Ligação entre 2.A e 2.B	Requisito Não-Funcional
Ligação entre 3.A e 3.C	Evento
3.J	Entidade
3.D	Operação
3.L	Refinamento E
3.K	Refinamento Ou
Ligação entre 3.A e 3.B	Conflito
	Obstáculo
	Relação de Obstrução
Ligação entre 3.L e 3.D	Relação de Solução
	Agente
	Agregação
Ligação entre 3.I e 3.J	Herança

- Descreva textualmente o que é representado por cada uma das figuras:

  Figura 1:

  Figura 2:

  Figura 3:

A.3 Resolução de problemas

Considere o problema do parque de estacionamento tratado na disciplina de ERDS. Crie o modelo de objectivos da situação "Abastecer veículo".

"(...)Para abastecer (e.g. 95 s/ chumbo, gasóleo) basta também que o identificador colado no pára-brisas do veículo seja válido. Um veículo aderente, ao aproximar-se de uma das várias bombas de abastecimento automático, activa o sistema que acende uma luz verde. Para abastecer, o condutor deve inserir primeiro o PIN do cartão de débito associado ao identificador. O abastecimento termina quando o condutor coloca a mangueira de volta no suporte da bomba. O valor a pagar, que depende do tipo e quantidade de combustível seleccionado, é mostrado no display da bomba. Uma vez terminada a operação o veículo pode seguir. O débito em conta é automático, portanto se a conta não tiver saldo suficiente o abastecimento não é autorizado.(...)".

- Crie o modelo de responsabilidades do Veículo e do Controlo de Entrada do Parque.

- Crie o modelo de objectos do Parque de Estacionamento.

- Para cada um dos modelos criados, omita alguns elementos dos diagramas criados (por exemplo, no modelo de objectivos experimente omitir os requisitos não-funcionais e/ou obstáculos).

A.4 Nível de satisfação

- Qual a adequação da sintaxe da linguagem à metodologia?

  ____ ____ ____ ____ ____
  Muito Adequada Razoável Nada Adequada

- Com que facilidade interpretou os modelos apresentados na secção A.2?

  ____ ____ ____ ____ ____
  Muita Facilidade Razoavelmente Muita Dificuldade

- Em comparação com outras ferramentas usadas para criar modelos KAOS, achou estes modelos mais simples ou mais difíceis de interpretar?

  ____ ____ ____ ____ ____
  Mais Simples Igual Mais Difícil

- Com que facilidade criou os modelos pedidos na secção A.3?

  ____ ____ ____ ____ ____
  Muita Facilidade Razoavelmente Muita Dificuldade

- Em comparação com outras ferramentas usadas para criar modelos KAOS, achou estes modelos mais simples ou mais difíceis de criar?

  ____ ____ ____ ____ ____
  Mais Simples Igual Mais Difícil

- Em comparação com outras ferramentas usadas para criar modelos KAOS, como acha que esta ferramenta lida com o problema de escalabilidade dos modelos?

  ____ ____ ____ ____ ____
  Melhor Igual Pior

- Em comparação com outras ferramentas usadas para criar modelos KAOS, como acha desta ferramenta ao nível de eficiência na criação dos modelos?

  ____ ____ ____ ____ ____
  Melhor Igual Pior

- Considera que esta ferramenta trouxe alguma inovação à metodologia KAOS?

  ____ ____ ____ ____ ____
  Nenhuma Alguma Bastante

Quais são, na sua opinião,os pontos fortes desta ferramenta?

Quais são os pontos fracos desta ferramenta?

Que inovações considera que a ferramenta trouxe?

O que sugere para que esta ferramenta possa ser melhorada?

B . Manual de utilizador da LED sobre a ferramenta de modelação para a metodologia KAOS

1. Instalação da ferramenta
   Para utilizar a ferramenta é necessário ter instalado o Eclipse e os plugins EMF e GMF. É possível obter o Eclipse em http://www.eclipse.org/downloads/. Após instalação do mesmo, podem-se obter os plugins realizando os seguintes passos:

   - No Eclipse ir a Help -> Software Updates -> Find and Install. Depois:
     - Search for new features to install
     - Calisto Discovery Site
   - Escolher então os seguintes components:
     - Eclipse Modeling Framework (EMF);
     - Graphical Modeling Framework (GMF).

2. Instalar e executar a LED KAOS
   Antes de fazer a instalação da LED, é necessário certificar que a versão do compilador Java é a 6.0. Para isso vai-se a Window -> Preferences -> Java -> Compiler -> Compiler compliance level. Após este passo, e a instalação do EMF/GMF, faz-se a instalação da LED. Para isso vai-se a File -> Import. Em Import escolher General -> Existing Projects into Workspace. Em Select archive file -> Browse indicar o caminho para o ficheiro KAOS.zip, seleccionam-se as pastas correspondentes e carrega-se em Finish (figura B.1).

3. Criação do Projecto
   Para se criar um projecto vai-se a File -> New -> Project. Na janela que aparece (figura B.2) ir a General -> Project. Carregar em Next, escrever o nome do projecto e carregar em Finish (figura B.3).
   Após a criação do projecto, deve-se criar o espaço de edição dos modelos. Para isso vai-se a File -> New -> Example e selecciona-se KAOSDiagram (figura B.4). Carrega-se em Next, escreve-se o nome pretendido e carrega-se em Finish (figura B.5). No projecto criado anteriormente aparecem dois ficheiros, um com extensão *.kaosstandard* e outro com a extensão *.kaosstandard_diagram*. O que vai ser usado na criação de modelos é o segundo.

4. Criação de Modelos
   Nesta secção apresenta-se o modo de utilização do editor da ferramenta.

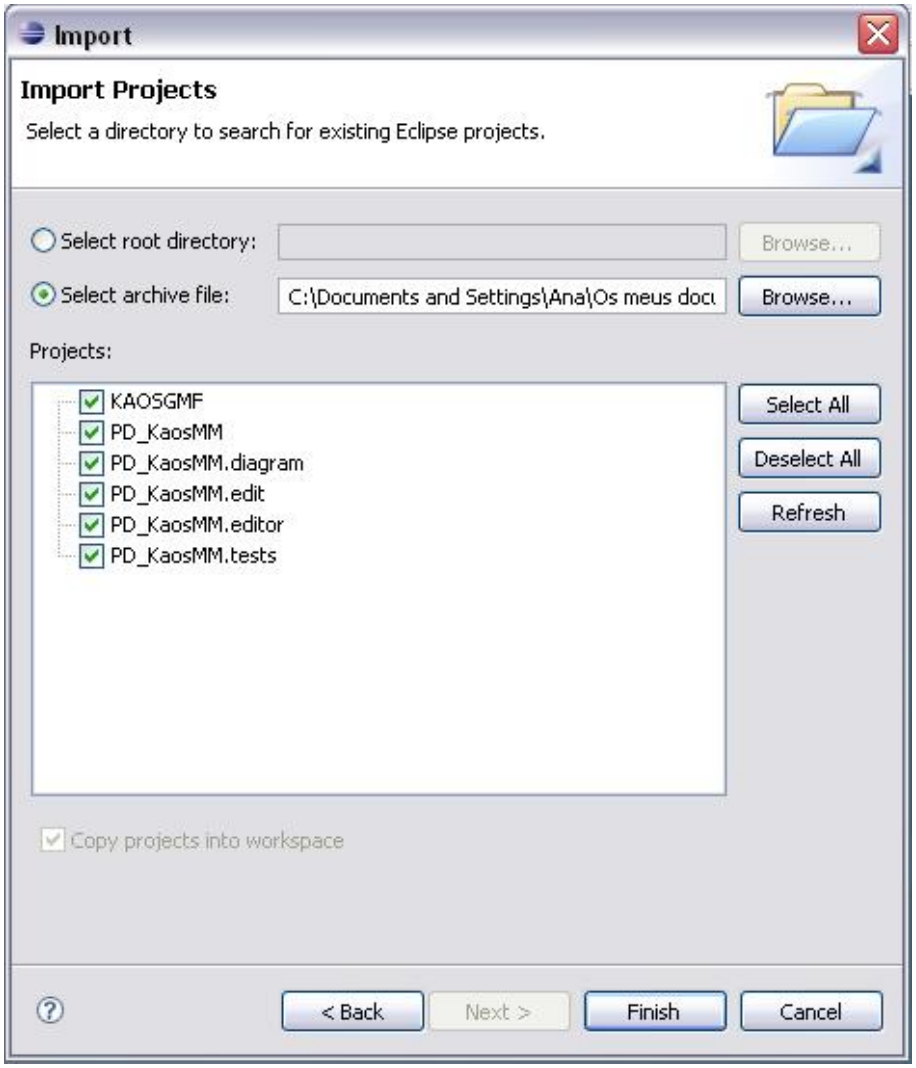

Figura B.1 Selecção de pastas para a LED KAOS.

- Menu de selecção
  A ferramenta apresenta no lado direito um menu de selecção, onde são mostrados os elementos necessários para construir os diagramas (figuras B.6 e B.7). O menu de selecção apresenta três tipos de elementos: *Compartments* (onde estão representados os compartimentos onde são armazenados os elementos dos diagramas), *Links* (as ligações entre os diferentes elementos) e *Nodes* (elementos que representam os conceitos do KAOS.
  Para realizar qualquer tipo de diagrama é necessário criar primeiro um compartimento, conforme o modelo que se quer criar. Como exemplo, vai ser criado um Modelo de Objectivos. Vão então ser realizados os seguintes passos:

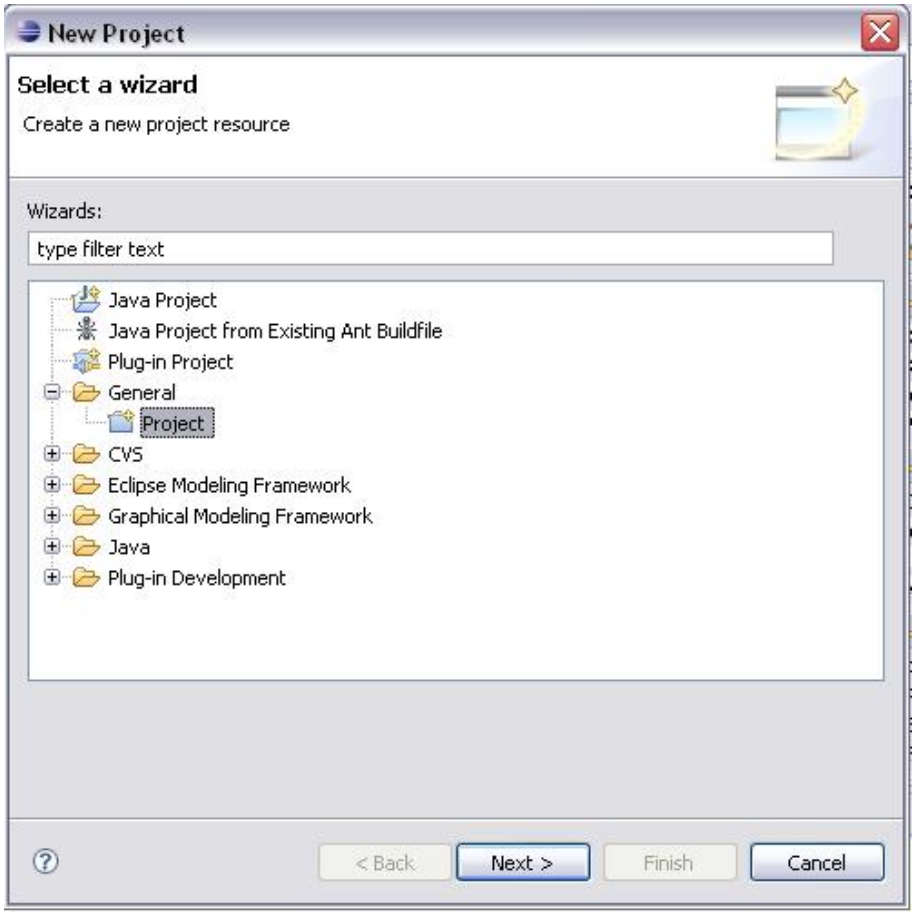

Figura B.2 Criação de Projecto (parte 1).

(a) Criar *Goal Compartments*: Para se poderem criar objectivos no modelo é necessário ter pelo menos um *Goal Compartment* (figura B.8). Existem dois meios de criação: (i) ir ao menu de selecção e escolher o elemento *GoalCompartment* (assinalado com a elipse vermelha); (ii) na janela de edição, quando aparecem quais os elementos disponíveis (elipse verde) escolher o elemento (neste caso, será GC - *GoalCompartment*).

(b) Criar Objectivos, Expectativas ou Requisitos: existem duas maneiras de criar estes elementos (figura B.9): (i) ir à secção *Nodes* e arrastar o elemento desejado para dentro do compartimento (elipse vermelha); (ii) dentro do compartimento, ao aparecer a lista dos elementos possíveis de criar dentro do compartimento, escolher o elemento desejado (elipse verde).

(c) Criar as ligações entre os elementos: Para se criar ligações entre os elementos,

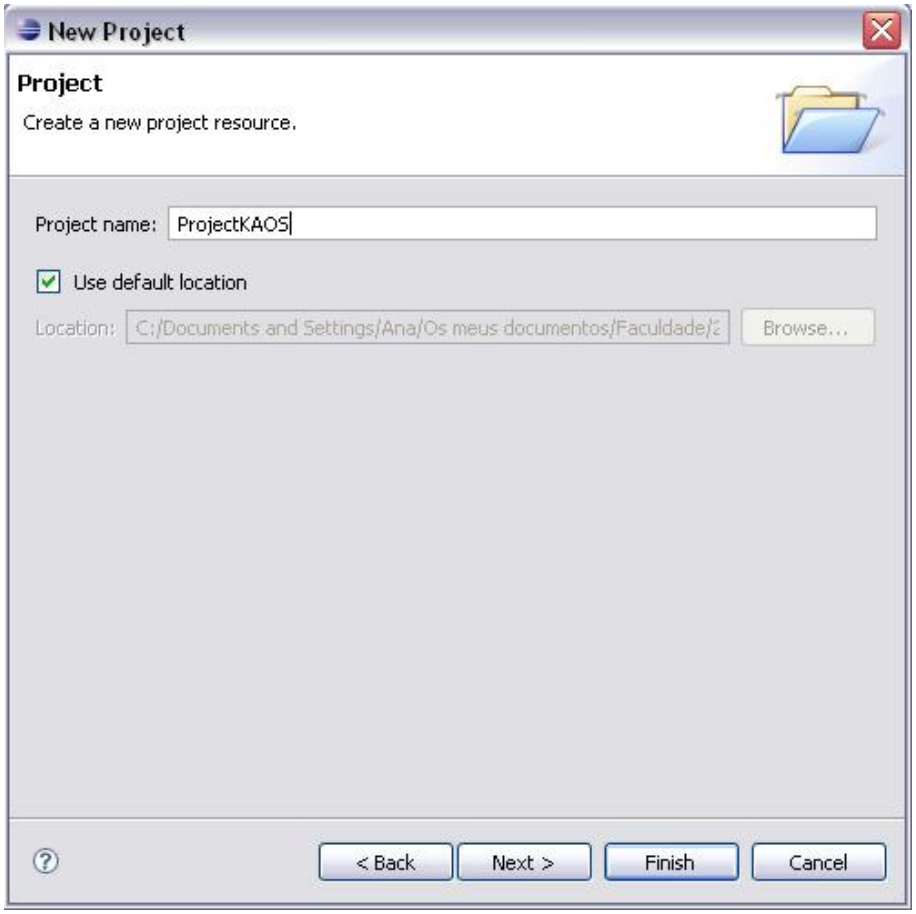

Figura B.3 Criação de Projecto (parte 2).

vai-se à secção *Links* (elipse vermelha) e escolhe-se a ligação desejada. Neste caso foi criado um *AndRefinement* (figura B.10).

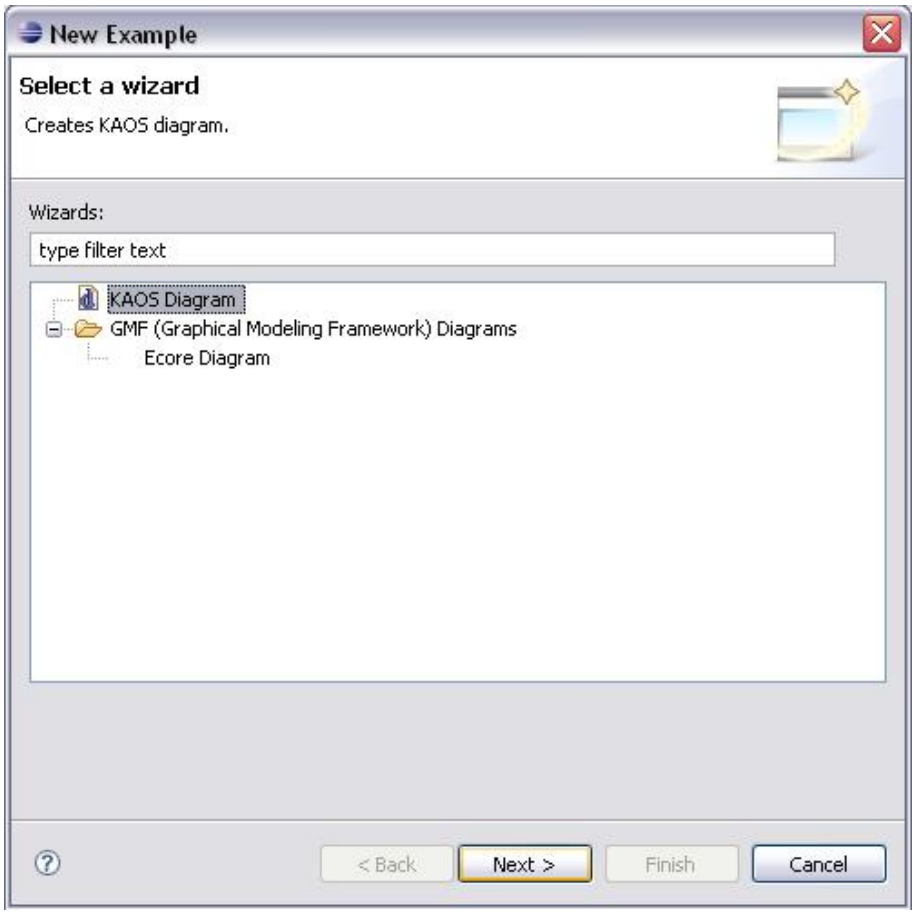

Figura B.4 Criação de Diagramas (parte 1).

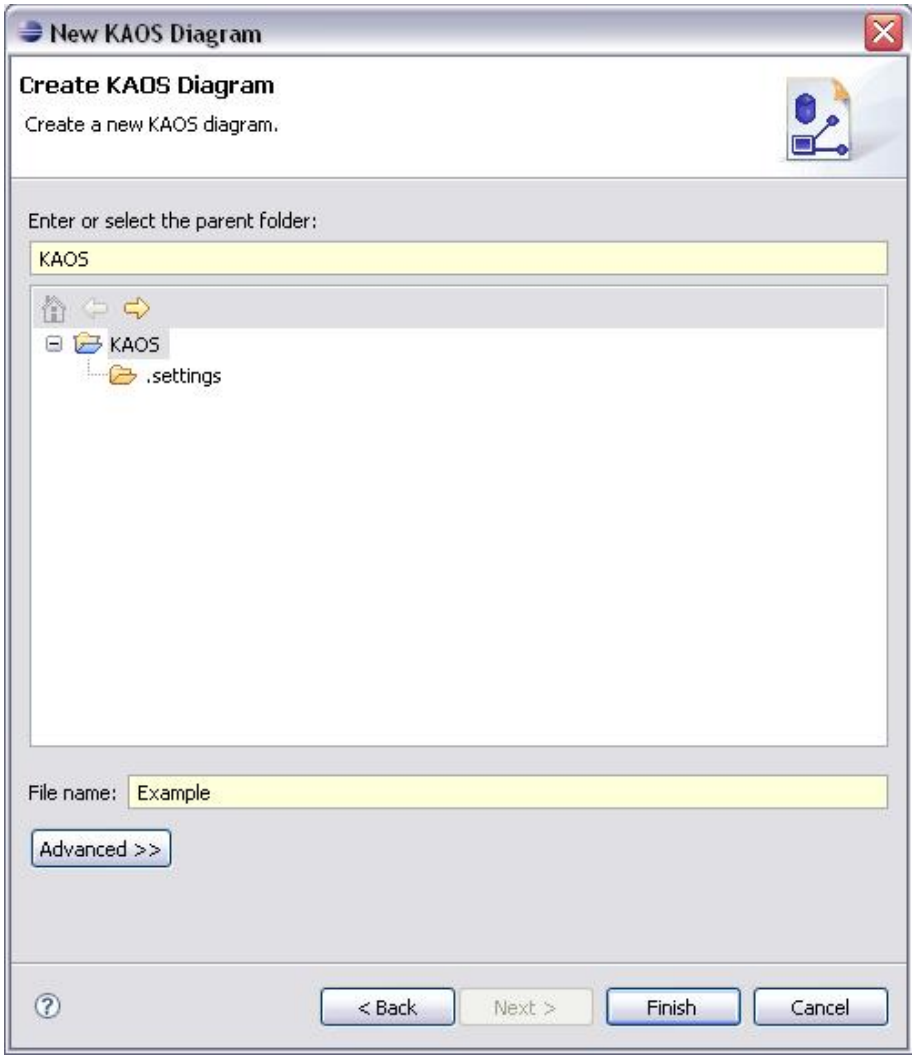

Figura B.5 Criação de Diagramas (parte 2).

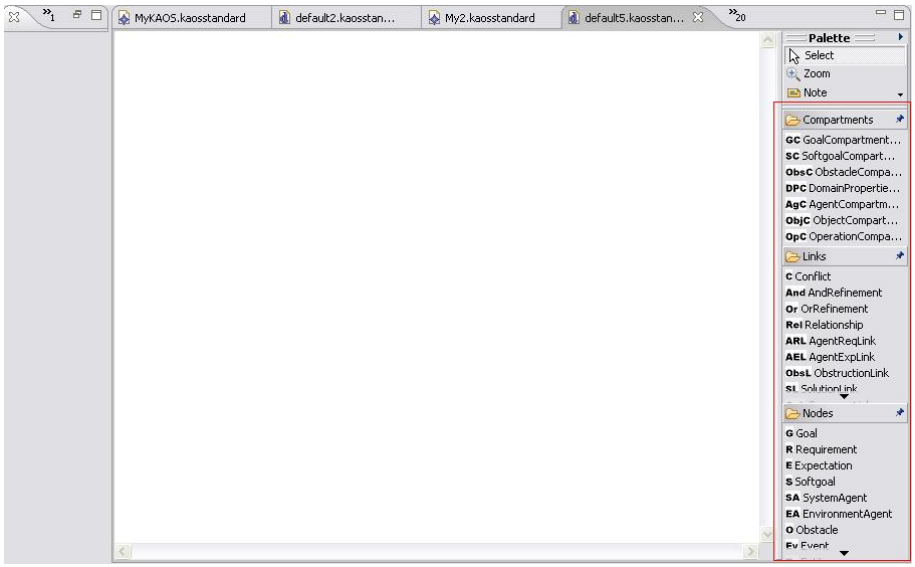

Figura B.6 Editor da ferramenta.

Figura B.7 Menu de selecção.

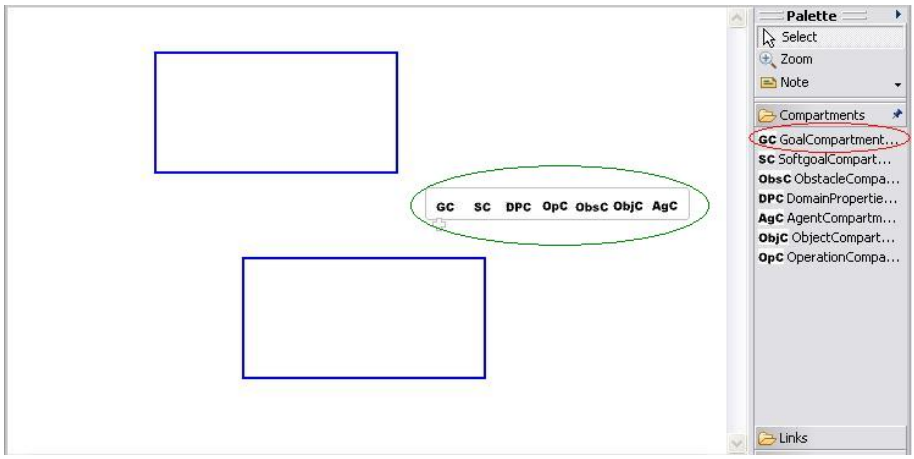

Figura B.8 Criação de Goal Compartments.

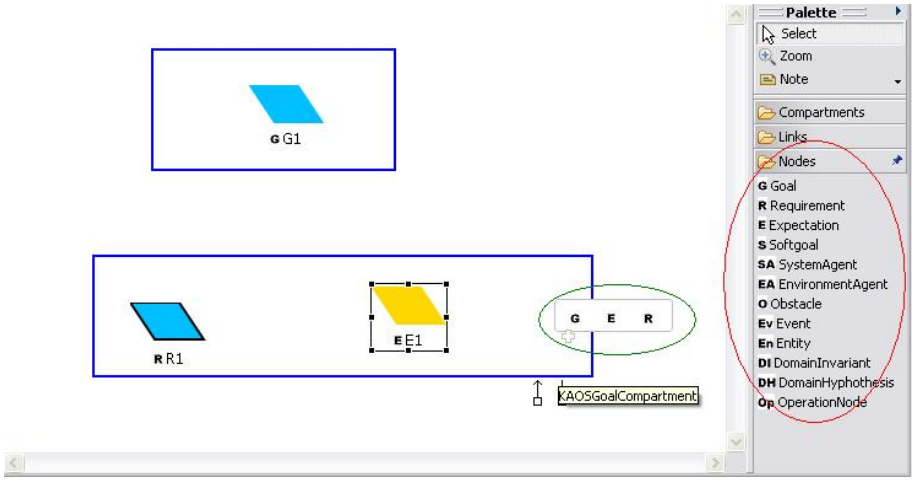

Figura B.9 Criar Objectivos, Requisitos ou Expectativas.

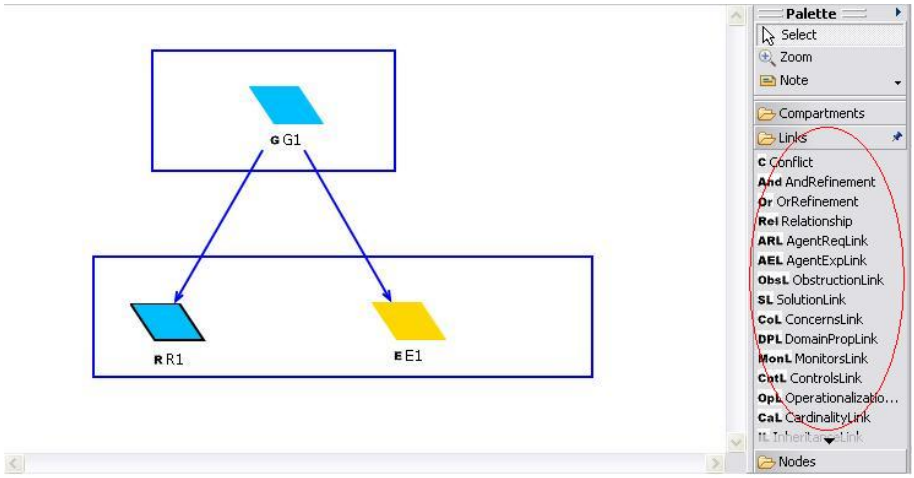

Figura B.10 Criação de ligações entre elementos.

Bibliografia

[1] Concern (computer science). Página Web. http://en.wikipedia.org/wiki/Concern_(computer_science).

[2] Dia a drawing program. Página Web. http://www.gnome.org/projects/dia/.

[3] Domain-specific language. Página Web. http://en.wikipedia.org/wiki/Domain_Specific_Languages.

[4] Eclipse Modeling - EMF - Home. Página Web. http://www.eclipse.org/modeling/emf/.

[5] Eclipse Tools - GEF Project. Página Web. http://www.eclipse.org/gef/overview.html.

[6] Emfatic - Eclipsepedia. http://wiki.eclipse.org/Emfatic. Wiki do Eclipse sobre o Emfatic.

[7] GME: The Generic Modeling Environment. Página Web. http://www.isis.vanderbilt.edu/projects/gme/.

[8] Graphical Modeling Framework. Página Web. http://www.eclipse.org/modeling/gmf/.

[9] GRL. Página Web. http://www.cs.utoronto.ca/km/GRL/.

[10] Klasse Objecten - Introduction to OCL. World Wide Web electronic publication. http://www.klasse.nl/ocl/ocl-introduction.html.

[11] Objectiver: HomePage. Página Web. http://www.objectiver.com/index.php?id=4.

[12] Organizational Modeling Environment (OME). http://www.cin.ufpe.br/ eti907/ferramentas/ome310j.zip. Link para download da ferramenta OME.

[13] Software Factories. Página Web. http://www.softwarefactories.com/.

[14] Third generation programming language. Página Web. http://en.wikipedia.org/wiki/Third-generation_programming_language.

[15] www.ample-project.net. http://www.ample-project.net/. Página do Projecto AMPLE. O caso de estudo encontra-se na secção “Documents”, “Work Package 5”, “Deliverable 5.2”.

[16] A. I. Antón e C. Potts. The Use of Goals to Surface Requirements for Evolving Systems. In *Proceedings ICSE-98: 20th International Conference on Software Engineering*, Quioto, Japão, Abril 1998.

[17] H. Al-Subaie e T. Maibaum. Evaluating the Effectiveness of a Goal-Oriented Requirements Engineering Method. In *Fourth International Workshops on Comparative Evaluation in Requirements Engineering (CERE'06)*, Minneapolis/Saint Paul, Estados Unidos da América, Setembro 2006.

[18] V. Backvanski e P. Graff. Mastering Eclipse Modeling Framework. apresentação, 2005.

[19] J. Bézivin, G. Hillairet, F. Jouault, I. Kurtev e W. Piers. Bridging the MS/DSL Tools and the Eclipse Modeling Framework. In *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*, San Diego, Estados Unidos da América, 2005.

[20] CEDITI sa, Gosselies, Bélgica. *A KAOS Tutorial*, Setembro 2003.

[21] S. Cook, G. Jones, S. Kent e A. Wills. *Domain-Specific Development with Visual Studio DSL Tools*. Addison-Wesley, 2007.

[22] E. Letier and A. van Lamsweerde. KAOS in Action: The BART System. Technical report, Universidade Católica de Louvain, Bélgica, 2000.

[23] A. Hamie, F. Civello, J. Howse, S. Kent e R. Mitchell. Reflections on the Object Constraint Language. In *UML'98: Beyond the Notation - International Workshop*, Mulhouse, França, 1998.

[24] J. L. Hammond. Extending Frameworks with Domain-Specific Languages. *Embedded System Engineering*, 15.5, Junho 2007.

[25] J. Kasser. Object-Oriented Requirements Engineering and Management. In *Systems Engineering Test and Evaluation (SETE) Conference*, Canberra, Austrália, 2003.

[26] S. Kelly e J.-P. Tolvanen. *Domain-Specific Modeling*. John Wiley & Sons, Inc., 2008.

[27] L. Kolos-Mazuryk, P. van Eck e R. Wieringa. A Survey of Requirements Engineering Methods for Pervasive Services. Technical report, Centre for Telematics and Information Technology (CTIT), Universidade de Twente, Holanda, Março 2006.

[28] A. Lapouchnian. Goal-Oriented Requirements Engineering: An Overview of the Current Research. Technical report, Universidade de Toronto, Canadá, 2005.

[29] E. Letier. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD Thesis, Universidade Católica de Louvain, Bélgica, Maio 2001.

[30] E. Letier, J. Kramer, J. Magee e S. Uchitel. Deriving event-based transition systems from goal-oriented requirements models. *Automated Software Engineering*, 15:175–206, 2008.

[31] R. Matulevicius e P. Heymans. Analysis of KAOS Meta-model. Technical report, Universidade de Namur, Bélgica, 2005.

[32] R. Matulevicius e P. Heymans. KAOS Construct Analysis using the UEML Approach Template. Technical report, Universidade de Namur, Bélgica, 2005.

[33] N. Murray, N. W. Paton, C. A. Goble and J. Bryce. Kaleidoquery - A Flow-based Visual Language and its Evaluation. *Journal of Visual Languages and Computing*, pages 151–189, 2000.

[34] R. Pressman. *Software Engineering - A Practitioner's Approach*. McGraw-Hill International Edition, 6ª edition, 2005.

[35] A. Rashid, P. Sawyer, A. Moreira e J. Araújo. Early Aspects: a Model for Aspect-Oriented Requirements Engineering. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02)*, Essen, Alemanha, Setembro 2002.

[36] I. Sommerville e P. Sawyer. Viewpoints: Principles, Problems and a Practical Approach to Requirements Engineering. Technical report, Cooperative Systems Engineering Group, Computing Department, Universidade de Lancaster, Reino Unido, 1997.

[37] V. Winter and R. Berg and J. Ringland. Bay Area Rapid Transit District Advance Automated Train Control System Case Study Description. Descrição informal do sistema BART por membros do Sandia National Laboratories.

[38] A. van Lammsweerde e E. Letier. Handling Obstacles in Goal-Driven Requirements Engineering. In *Proceedings of ICSE'98 - 20th International Conference on Software Engineering*, 1998.

[39] A. van Lamsweerde. Goal-Oriented Requirements Enginnering: A Guided Tour, Agosto 2001. Artigo Convidado para RE'01 - 5th IEEE International Symposium on Requirements Engineering.

[40] A. van Lamsweerde. Goal-Oriented Requirements Engineering: from System Objectives to UML Models to Precise Software Specifications. apresentação, 2003.

[41] A. van Lamsweerde, R. Darimont e E. Letier. Managing Conflicts in Goal-Driven Requirements Engineering. In *IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development*, Novembro 1998.

[42] A. van Lamsweerde e E. Letier. From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering. In Springer-Verlag LNCS, editor, *Radical Innovations of Software & Systems Engineering, Post-Workshop Proceedings of the Monterey '02 Workshop*, 2003.

[43] M. Vaziri e D. Jackson. Some Shortcomings of OCL, the Object Constraint Language of UML. In *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, 2000.